Problem Statement

1. The Objective is to predict driver attrition at Ola by analyzing attributes like demographics, tenure, and historical performance data.
2. The biggest challenges are
    A. High churn rates among drivers.
    B. Competitive industry where drivers easily switch between Ola and Uber based on rates.
3. Recruitment Strategy: Ola's strategy includes casting a wide net, targeting individuals without cars for driving jobs. However, this strategy is deemed costly.
4. Impact of Churn:Frequent driver turnover negatively affects organizational morale.
5. Cost Considerations: Acquiring new drivers is more expensive than retaining existing ones.
6. As a data scientist, the primary task is to develop a predictive model for driver attrition based on provided attributes.
7. Addressing driver attrition is crucial for sustaining a motivated and efficient driver team amid industry competition and fostering organizational growth.

Import the dataset and do usual exploratory analysis steps like checking the structure & characteristics of the dataset.

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
#load the dataset
data = pd.read_csv("https://d2beiqkhq929f0.cloudfront.net/public_assets/assets/000/
```

In [4]:
```
data.shape
# dataset has 19104 records with 14 features
```

Out[4]:
```
(19104, 14)
```

In [5]:
```
data.head(2)
# The column Unnamed:0 is irrelvant column. Delete it.
```

Out[5]:

	Unnamed: 0	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	La
0	0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	
1	1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	

In [6]:
```
data.drop(columns = "Unnamed: 0",inplace = True)
```

In [7]:
```
data.head(2)
```

Out[7]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingD
0	01/01/19	1	28.0	0.0	C23	2	57387	24/12/18	N
1	02/01/19	1	28.0	0.0	C23	2	57387	24/12/18	N

In [8]:
```
data_copy = data.copy(deep = True)
```

In [9]:
```
data.info()
# 13 features are there.
# Some features have missing or null values
# Some features are not aligned with their assigned datatypes
# "MMM-YY" feature is representing date. The datatype needs to be changed.
# Similarly, datatype of features "Dateofjoining" and "LastWorkingDate" features ne
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   MMM-YY                19104 non-null  object
 1   Driver_ID             19104 non-null  int64
 2   Age                   19043 non-null  float64
 3   Gender                19052 non-null  float64
 4   City                  19104 non-null  object
 5   Education_Level       19104 non-null  int64
 6   Income                19104 non-null  int64
 7   Dateofjoining         19104 non-null  object
 8   LastWorkingDate       1616 non-null   object
 9   Joining Designation   19104 non-null  int64
 10  Grade                 19104 non-null  int64
 11  Total Business Value  19104 non-null  int64
 12  Quarterly Rating      19104 non-null  int64
dtypes: float64(2), int64(7), object(4)
memory usage: 1.9+ MB
```

Check for missing values

In [10]:
```
# count of missing values in column
display(data[data.columns[data.isnull().any()]].isnull().sum())
# percentage of missing values
display(data[data.columns[data.isnull().any()]].isnull().sum()*100/data.shape[0])

# Age has missing values which will be treated using imputation techniques later
# Missing values in LastWorkingDate feature tells whether employee has left or not.
# any imputation
# Gender will also be treated.
```

```
Age                   61
Gender                52
LastWorkingDate    17488
dtype: int64
```

```
Age                0.319305
Gender             0.272194
LastWorkingDate   91.541039
dtype: float64
```

Check for duplicate values

In [11]:
```python
data.duplicated().sum()
# There are no duplicate records in dataset
```

Out[11]:
```
0
```

Convert date-like features to their respective data type

In [12]:
```python
data = data.astype({'MMM-YY': np.datetime64,'Dateofjoining': np.datetime64})
# feature "LastWorkingdate" is also datetime type but as this feature will directly
# leave this feature as it is as of now
```

In [13]:
```python
data[["MMM-YY",'Dateofjoining']].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19104 entries, 0 to 19103
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   MMM-YY         19104 non-null  datetime64[ns]
 1   Dateofjoining  19104 non-null  datetime64[ns]
dtypes: datetime64[ns](2)
memory usage: 298.6 KB
```

Univariate Analysis

In [14]:
```python
from IPython.display import display, HTML
CSS = """
.output {
 flex-direction: row;
}
"""
HTML('<style>{}</style>'.format(CSS))
```

Out[14]:

Analysis of datetime features: MMM-YY, Date of Joining, Last Working Date

In [15]:
```python
# Detail analysis of all datetime features
display(data["MMM-YY"].value_counts())
display(data["Dateofjoining"].value_counts())
display(data["LastWorkingDate"].value_counts())
```

```
2019-01-01    1022
2019-02-01     944
2019-03-01     870
2020-12-01     819
2020-10-01     818
2020-08-01     812
2020-09-01     809
2020-07-01     806
2020-11-01     805
2019-12-01     795
2019-04-01     794
2020-01-01     782
2019-11-01     781
2020-06-01     770
2020-05-01     766
2019-05-01     764
2019-09-01     762
2020-02-01     761
2019-07-01     757
2019-08-01     754
2019-10-01     739
2020-04-01     729
2019-06-01     726
2020-03-01     719
Name: MMM-YY, dtype: int64
2015-07-23    192
2020-07-31    150
2019-07-04    146
2016-04-25    134
2015-07-30    118
             ...
2018-03-16      1
2018-09-26      1
2020-12-27      1
2018-12-29      1
2018-12-16      1
Name: Dateofjoining, Length: 869, dtype: int64
29/07/20    70
22/09/19    26
17/03/19    14
28/11/20    13
17/02/20    13
            ..
16/06/19     1
17/11/20     1
12/05/20     1
09/02/19     1
28/10/20     1
Name: LastWorkingDate, Length: 493, dtype: int64
```

Analysis of Categorical feature: Gender, Education level, Joining Designation, Grade, Quaterly Rating and City

In [16]:

```
# detail analysis of categorical features
display(data["Gender"].value_counts())
display(data["Education_Level"].value_counts())
display(data["Joining Designation"].value_counts())
display(data["Grade"].value_counts())

# Gender : 0 represents Male and 1 represents female
# Education level - 0 for 10+ ,1 for 12+ ,2 for graduate
# Joining Designation : Designation of the driver at the time of joining
# Grade : Grade of the driver at the time of reporting time, likely denoting perfor
```

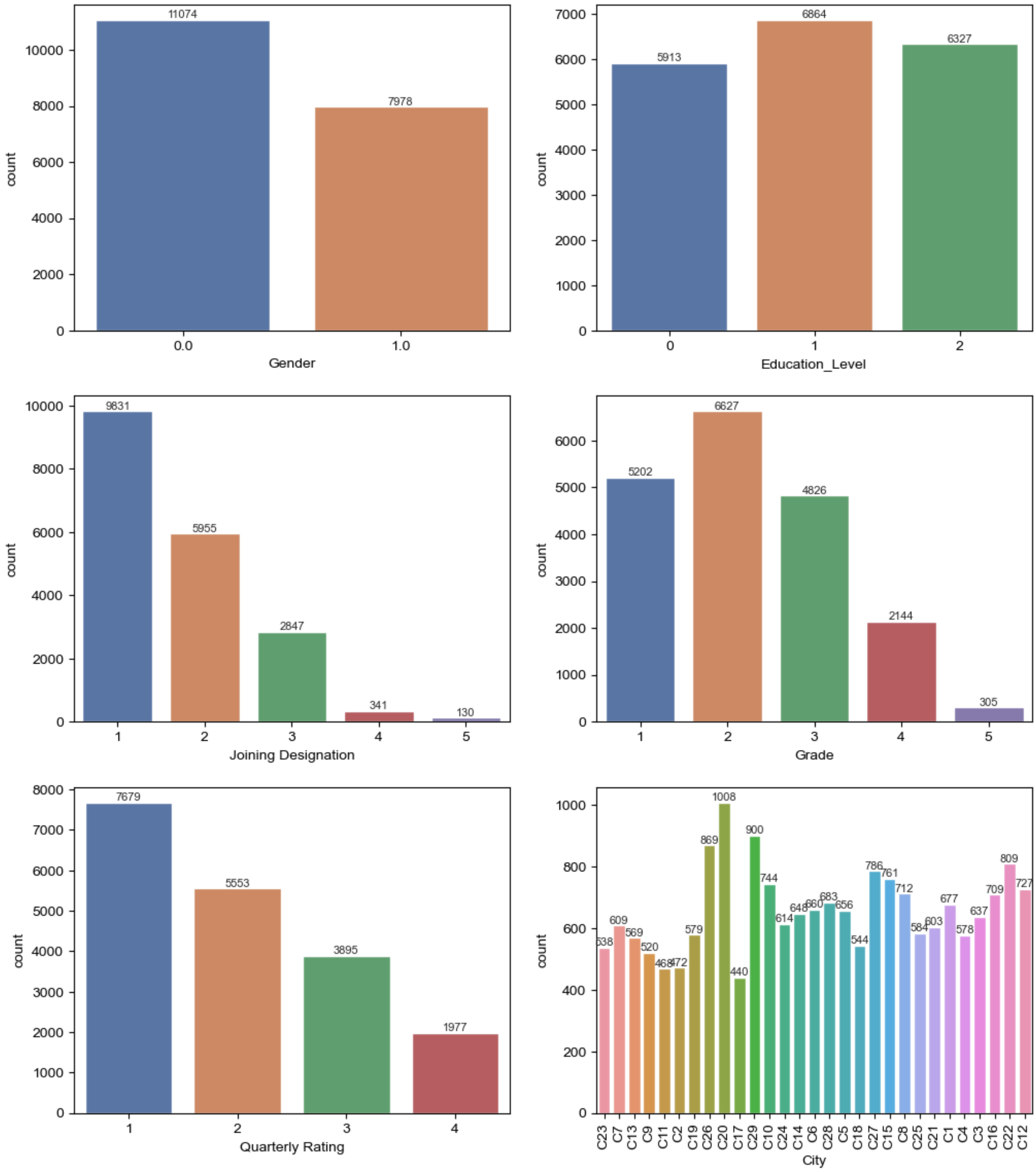
```
0.0    11074
1.0     7978
Name: Gender, dtype: int64
1    6864
2    6327
0    5913
Name: Education_Level, dtype: int64
1    9831
2    5955
3    2847
4     341
5     130
Name: Joining Designation, dtype: int64
2    6627
1    5202
3    4826
4    2144
5     305
Name: Grade, dtype: int64
```

In [17]:
```
display(data["Quarterly Rating"].value_counts())
display(data["City"].value_counts())

# Quarterly Rating : Quarterly rating of the driver: 1,2,3,4,5 (higher is better)
# City : City Code of the driver
```

```
1    7679
2    5553
3    3895
4    1977
Name: Quarterly Rating, dtype: int64
C20    1008
C29     900
C26     869
C22     809
C27     786
C15     761
C10     744
C12     727
C8      712
C16     709
C28     683
C1      677
C6      660
C5      656
C14     648
C3      637
C24     614
C7      609
C21     603
C25     584
C19     579
C4      578
C13     569
C18     544
C23     538
C9      520
C2      472
C11     468
C17     440
Name: City, dtype: int64
```

In [18]:
```
#Univariate analysis of categorical fields
cat_cols = ["Gender", "Education_Level", "Joining Designation", "Grade", "Quarterly
```

```
fig, axis = plt.subplots(nrows=3, ncols=2, figsize=(12, 14))
index = 0
for row in range(3):
    for col in range(2):
        sns.set(rc={'figure.figsize':(5,4)})
        ax = sns.countplot(data = data, x = data[cat_cols[index]], ax=axis[row, col
        ax.bar_label(container=ax.containers[0], fontsize=8)
        index += 1
plt.xticks(rotation=90)
plt.show()
```



In [19]:
```
# 1. What percentage of drivers have received a quarterly rating of 5 ?
data["Quarterly Rating"].value_counts()

# None has received rating of 5.
```

Out[19]:
```
1    7679
2    5553
3    3895
4    1977
Name: Quarterly Rating, dtype: int64
```

In [20]:
```
# 2. Comment on the correlation between Age and Quarterly Rating.?
data[["Age","Quarterly Rating"]].corr()

# The correlation value between Age and Quarterly Rating is 0.171632. This value is
# that is, there is very low correlation between these two features.
```

Out[20]:

	Age	Quarterly Rating
Age	1.000000	0.171818
Quarterly Rating	0.171818	1.000000

In [21]:
```
data[cat_cols].describe()

# 51% (9831 rows) of drivers join on Designation 1, followed by '2' with 31% (5955
# 34% (6627 rows) of drivers have Grade 2, followed by Grade '1' with 27% (5202 row
# 40% of drivers have Quarterly Rating 1, followed by Quarterly Rating '2' with 29%
# 58% of drivers are male, while 41% are female.
```

Out[21]:

	Gender	Education_Level	Joining Designation	Grade	Quarterly Rating
count	19052.000000	19104.000000	19104.000000	19104.000000	19104.000000
mean	0.418749	1.021671	1.690536	2.252670	2.008899
std	0.493367	0.800167	0.836984	1.026512	1.009832
min	0.000000	0.000000	1.000000	1.000000	1.000000
25%	0.000000	0.000000	1.000000	1.000000	1.000000
50%	0.000000	1.000000	1.000000	2.000000	2.000000
75%	1.000000	2.000000	2.000000	3.000000	3.000000
max	1.000000	2.000000	5.000000	5.000000	4.000000

In [22]:
```
data[cat_cols].describe(include = object)

# Total of 29 cities
# C20 city has highest drivers 1008, then city 'C29' with 900 drivers.
```

Out[22]:

	City
count	19104
unique	29
top	C20
freq	1008

Analysis of Numerical feature: Age, Income, Total Business Value

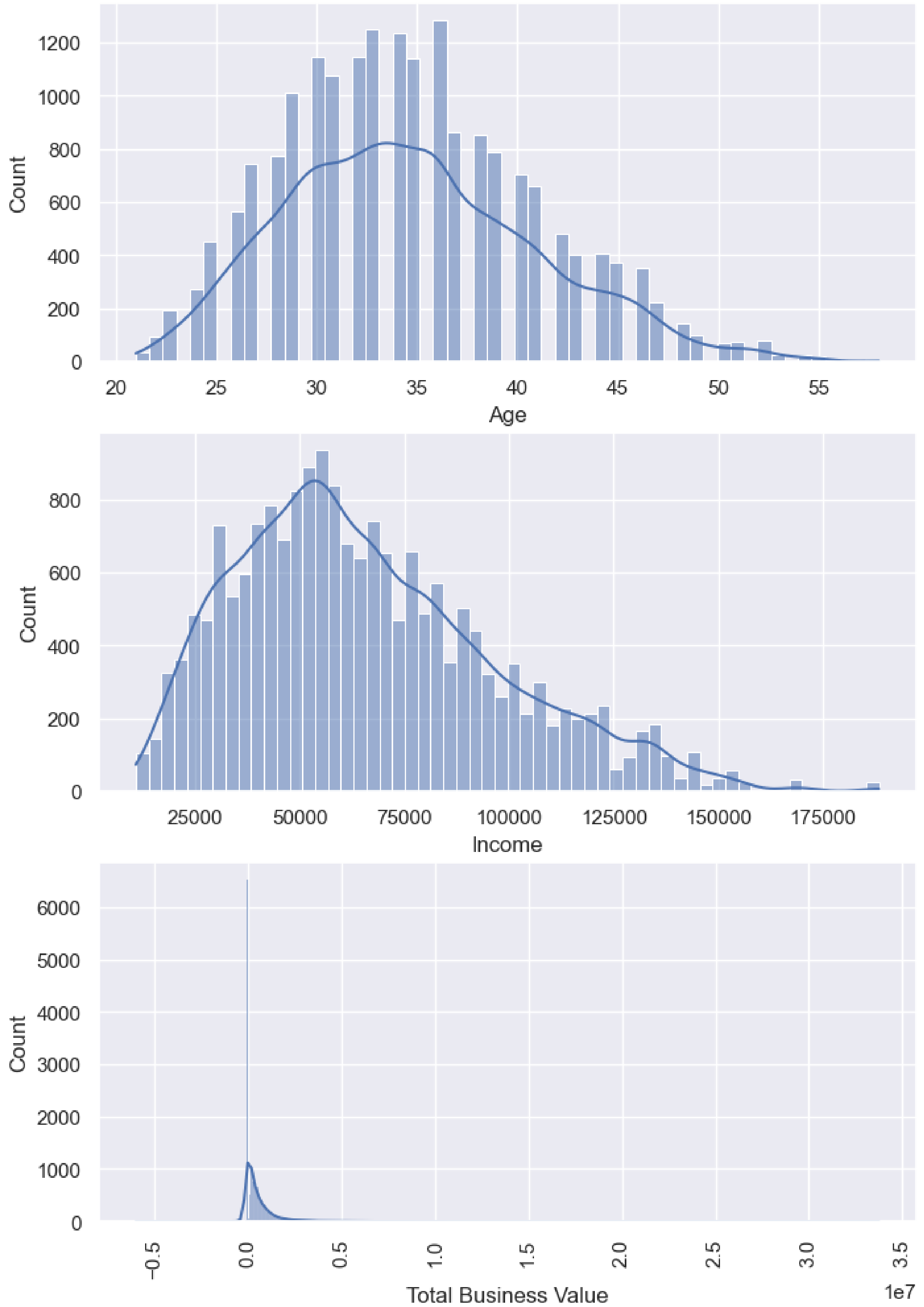
In [23]:
```
# detail analysis of numerical features
display(data["Age"].value_counts())
display(data["Income"].value_counts())
display(data["Total Business Value"].value_counts())
```

```
36.0    1283
33.0    1250
34.0    1234
30.0    1146
32.0    1143
35.0    1138
31.0    1076
29.0    1013
37.0     862
38.0     854
39.0     788
28.0     772
27.0     744
40.0     701
41.0     661
26.0     566
42.0     478
25.0     449
44.0     407
43.0     399
45.0     371
46.0     350
24.0     274
47.0     224
23.0     193
48.0     144
49.0      99
22.0      92
52.0      78
51.0      72
50.0      69
21.0      35
53.0      26
54.0      24
55.0      21
58.0       7
Name: Age, dtype: int64
48747     57
109652    32
68356     30
42260     28
67490     28
          ..
44706      1
72186      1
67162      1
22132      1
35091      1
Name: Income, Length: 2383, dtype: int64
0         6499
200000     288
250000     148
500000     131
300000     107
          ...
130520       1
275330       1
820160       1
203040       1
448370       1
Name: Total Business Value, Length: 10181, dtype: int64
```

In [24]:

```
#Univariate analysis of numerical fields
num_cols = ["Age", "Income", "Total Business Value"]
```

```
fig, axis = plt.subplots(nrows=3, ncols=1, figsize=(8, 12))
index = 0
for row in range(3):
    sns.histplot(data[num_cols[index]], ax=axis[row], kde = True, palette="bright")
    index += 1
plt.xticks(rotation=90)
plt.show()
```



In [25]:

```
data[num_cols].describe()

# Income is slight right skewed. Min income is 10747, Max is 188418. Average income
```

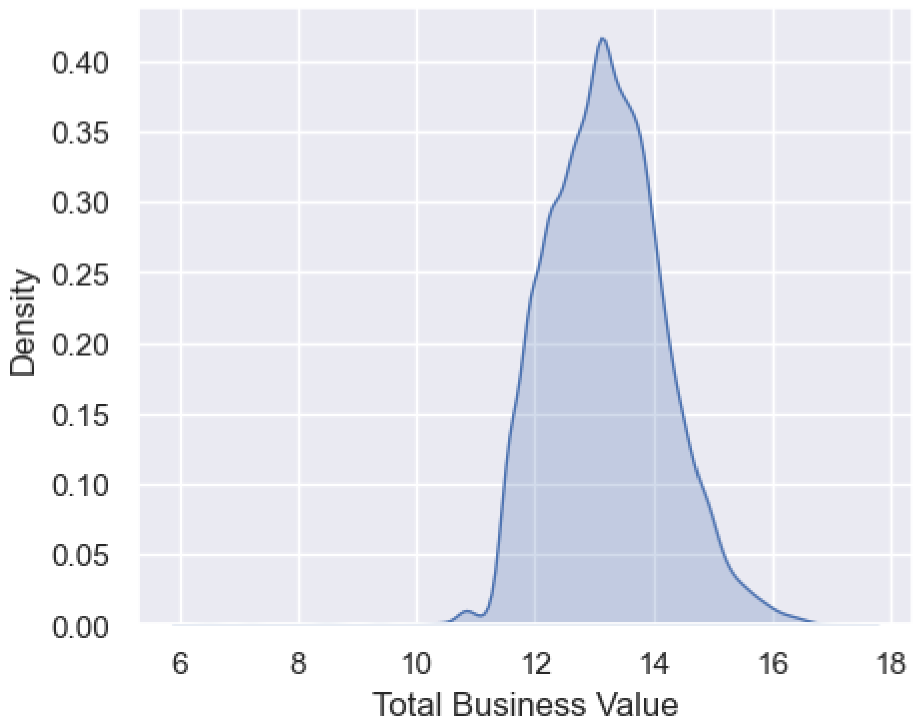
```
# Total Business Value is slight right skewed. Min value is -6.000000e+06 , Max val
# Average Value is 5.716621e+05
```

Out[25]:

	Age	Income	Total Business Value
count	19043.000000	19104.000000	1.910400e+04
mean	34.668435	65652.025126	5.716621e+05
std	6.257912	30914.515344	1.128312e+06
min	21.000000	10747.000000	-6.000000e+06
25%	30.000000	42383.000000	0.000000e+00
50%	34.000000	60087.000000	2.500000e+05
75%	39.000000	83969.000000	6.997000e+05
max	58.000000	188418.000000	3.374772e+07

In [26]:

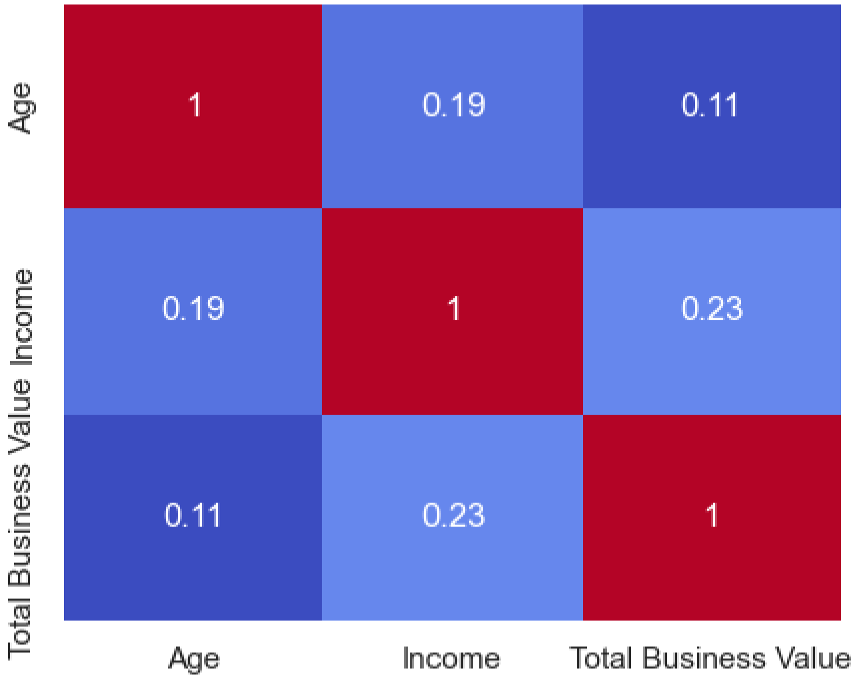
```
# As the variable " Total Business Value" is large values we will take log to check
sns.kdeplot(np.log(data['Total Business Value']),shade=True)
plt.show()
# Log distribution seems to follow slight normal shape.
```



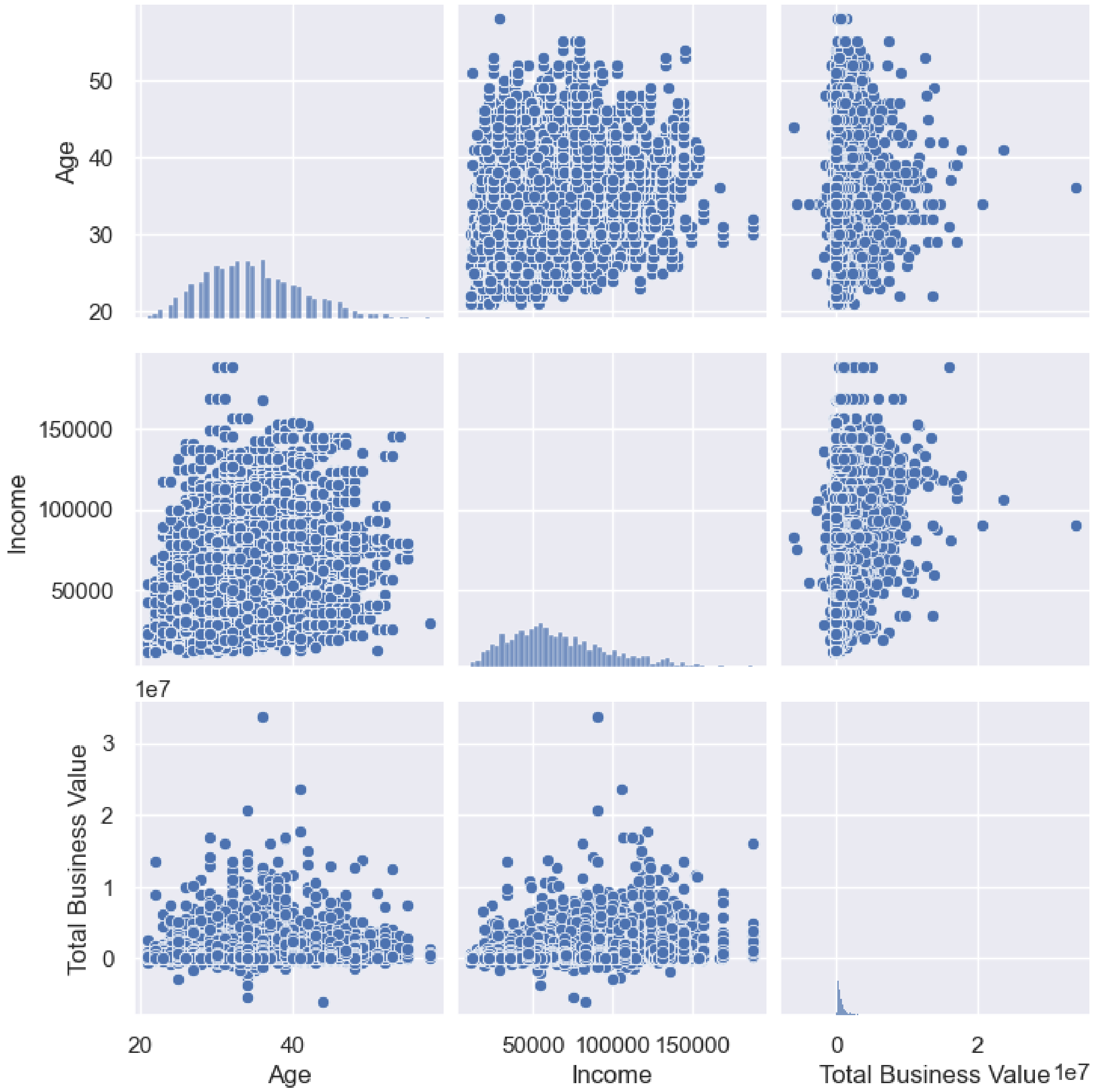
Bivariate Analysis

In [27]:

```
# Correlation among numerical features
sns.heatmap(data[num_cols].corr(),annot=True, cmap="coolwarm", cbar=False)
plt.show()

# there is very less correlation between all numerical features
```

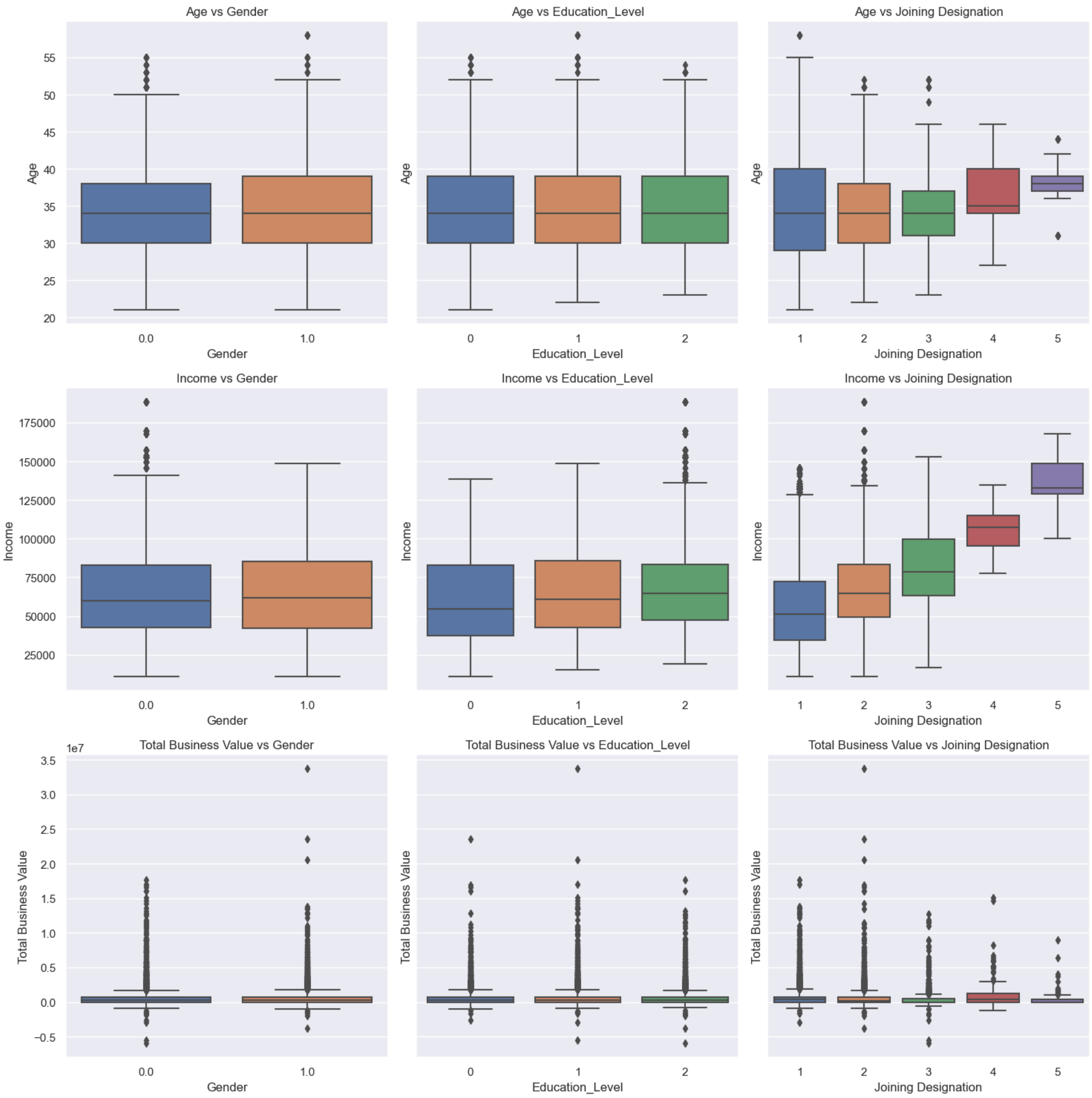


In [28]:
```
# plot pairplot for numerical features
sns.pairplot(data[num_cols])
plt.show()
```



In [29]:

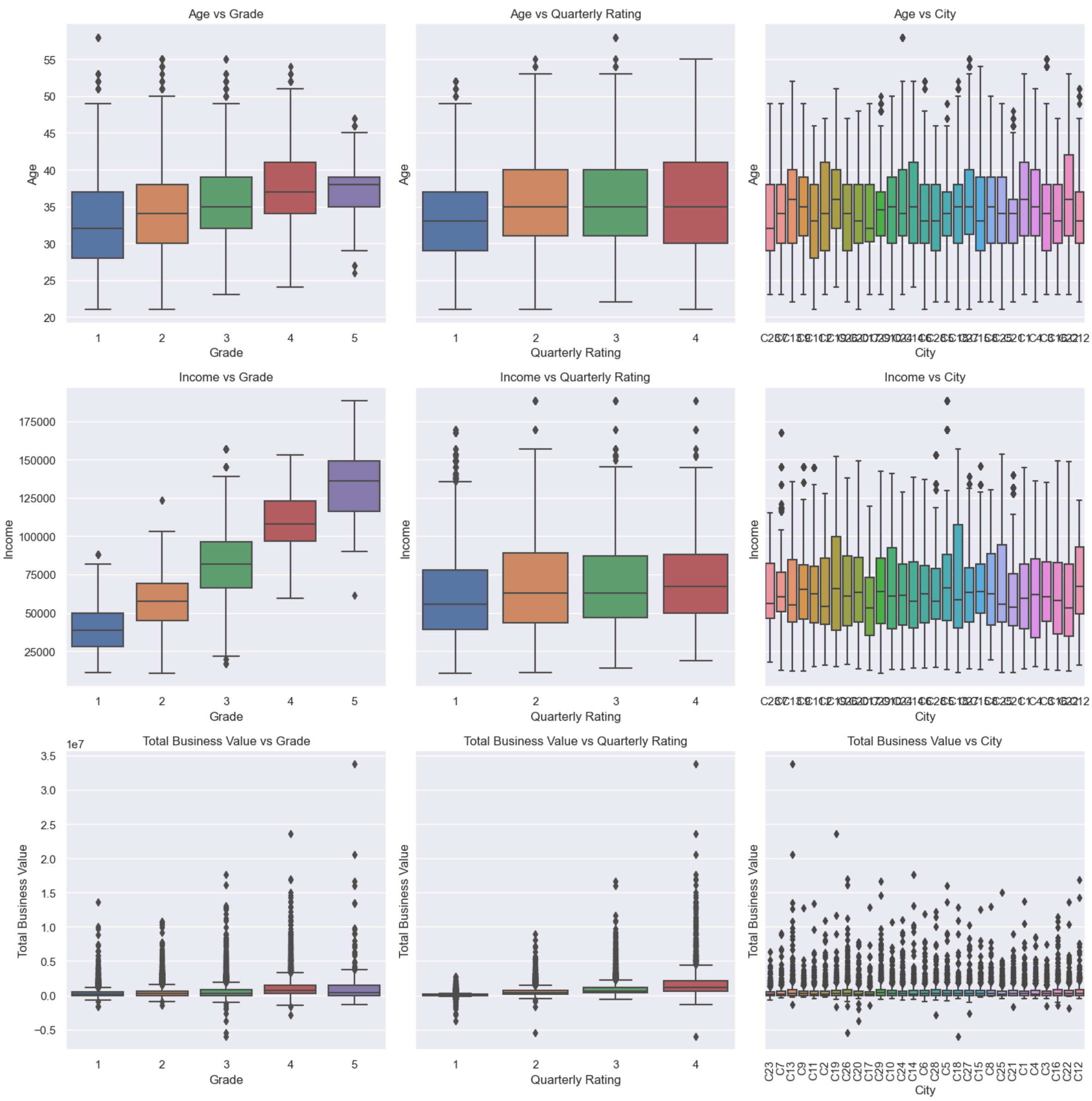
```
# Check dependency of all numerical features with repect to all categorical feature
num_cols = ["Age","Income", "Total Business Value"]
cat_cols_1 = cat_cols[:3]
data_long_1 = pd.melt(data, id_vars=cat_cols_1, value_vars=num_cols)

fig, axes = plt.subplots(nrows=len(num_cols), ncols=len(cat_cols_1), figsize=(15, 1

for i, num_col in enumerate(num_cols):
    for j, cat_col in enumerate(cat_cols_1):
        sns.boxplot(x=cat_col, y=num_col, data=data, ax=axes[i, j])
        axes[i, j].set_title(f'{num_col} vs {cat_col}')

plt.tight_layout()
plt.show()
```



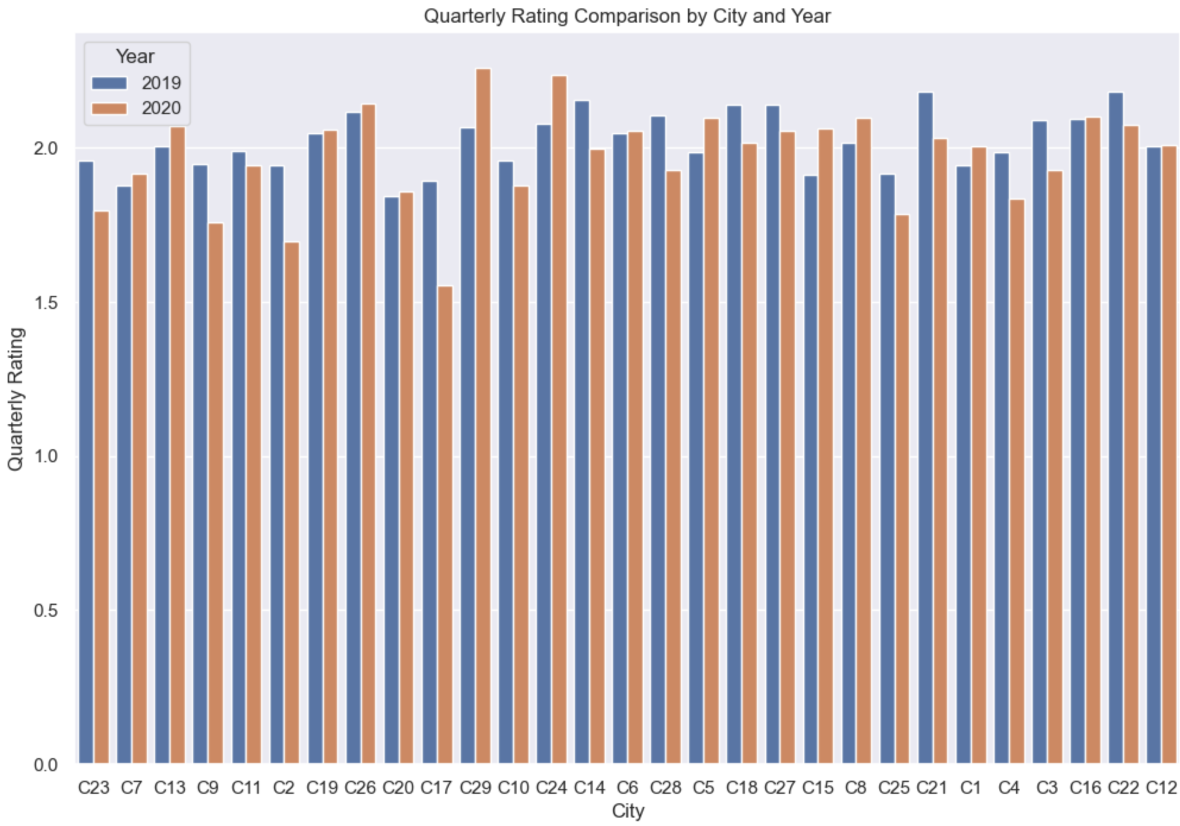
In [30]:

```
cat_cols_2 = cat_cols[3:]
data_long_2 = pd.melt(data, id_vars=cat_cols_2, value_vars=num_cols)

fig, axes = plt.subplots(nrows=len(num_cols), ncols=len(cat_cols_2), figsize=(15, 1

for i, num_col in enumerate(num_cols):
    for j, cat_col in enumerate(cat_cols_2):
        sns.boxplot(x=cat_col, y=num_col, data=data, ax=axes[i, j])
        axes[i, j].set_title(f'{num_col} vs {cat_col}')
plt.xticks(rotation=90)
```

```python
plt.tight_layout()
plt.show()
```



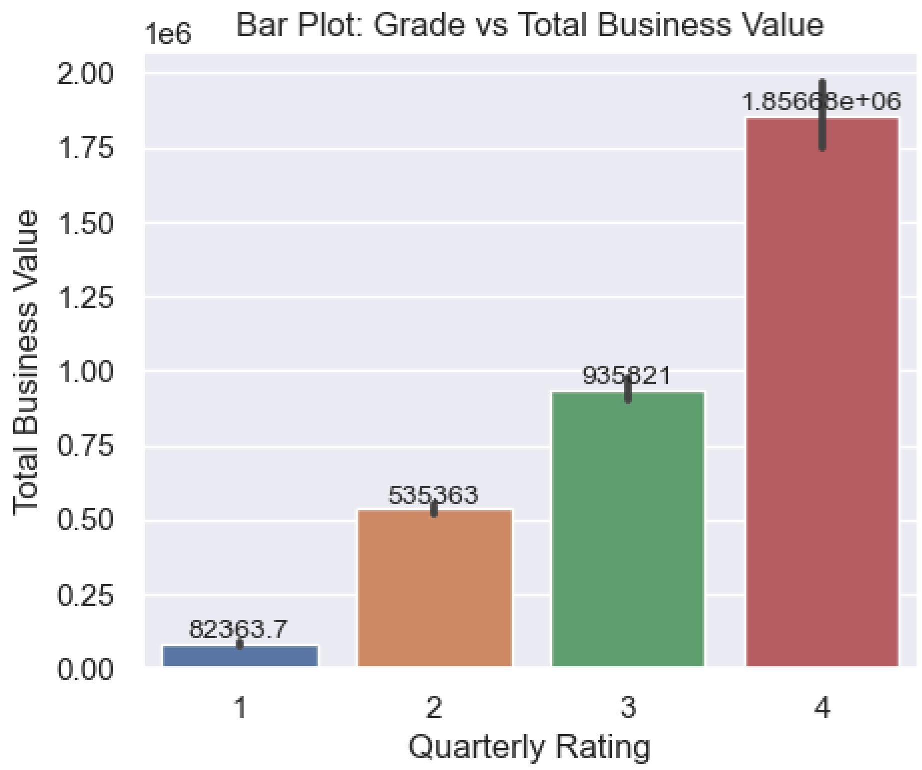
In [31]:
```python
# 3. Name the city which showed the most improvement in Quarterly Rating over the p

q3 = data[["City", "Quarterly Rating"]]
q3['Year'] = data['MMM-YY'].dt.year
q3
# Create a grouped bar plot using Seaborn
plt.figure(figsize=(12, 8))
sns.barplot(x='City', y='Quarterly Rating', hue='Year', data=q3, ci=None)

# Add labels and title
plt.xlabel('City')
plt.ylabel('Quarterly Rating')
plt.title('Quarterly Rating Comparison by City and Year')
plt.show()

# C29 is showing highest improvements in quaterly rating among all cities over past
```



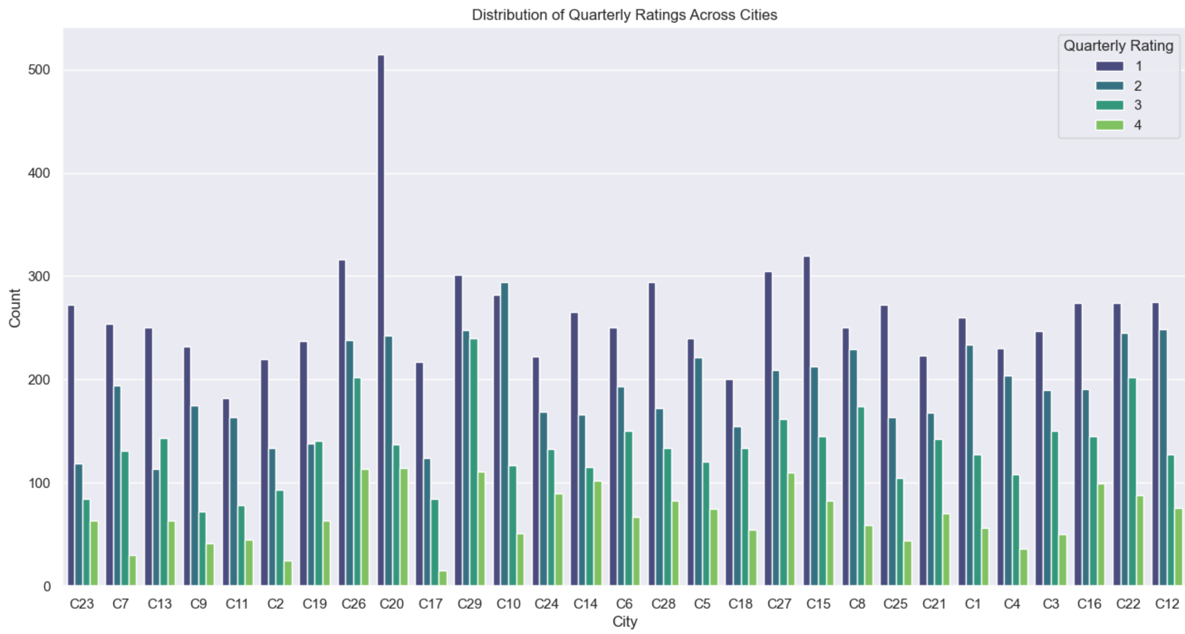
In [32]:

```
# 4. Drivers with a Grade of ‘A’ are more likely to have a higher Total Business Va
sns.barplot(x='Grade', y='Total Business Value', data=data)

# Add labels and title
plt.xlabel('Grade')
plt.ylabel('Total Business Value')
plt.title('Bar Plot: Grade vs Total Business Value')
plt.show()

# Grade with 4 is more likely to have higher Total Business Value
# Grade are represented as 1,2,3,4, and 5. If 1 represents as A, then above stateme
# If 5 represents as A, then also above statement is wrong. Therefore, the statemen
```

In [33]:
```
# 5. If a driver's Quarterly Rating drops significantly, how does it impact their T
# Business Value in the subsequent period?
ax = sns.barplot(x='Quarterly Rating', y='Total Business Value', data=data)
ax.bar_label(container=ax.containers[0], fontsize=10)
# Add labels and title
plt.xlabel('Quarterly Rating')
plt.ylabel('Total Business Value')
plt.title('Bar Plot: Grade vs Total Business Value')

plt.show()

# for rating of 4, the total business value is approx 1.115*10^6.
# for rating of 3, total business value is 767804
# for rating of 2, total business value is 494266
# for rating of 1, total business value is 83102.9
# It can be observed that as rating decreases by unity, total business value drasti
```

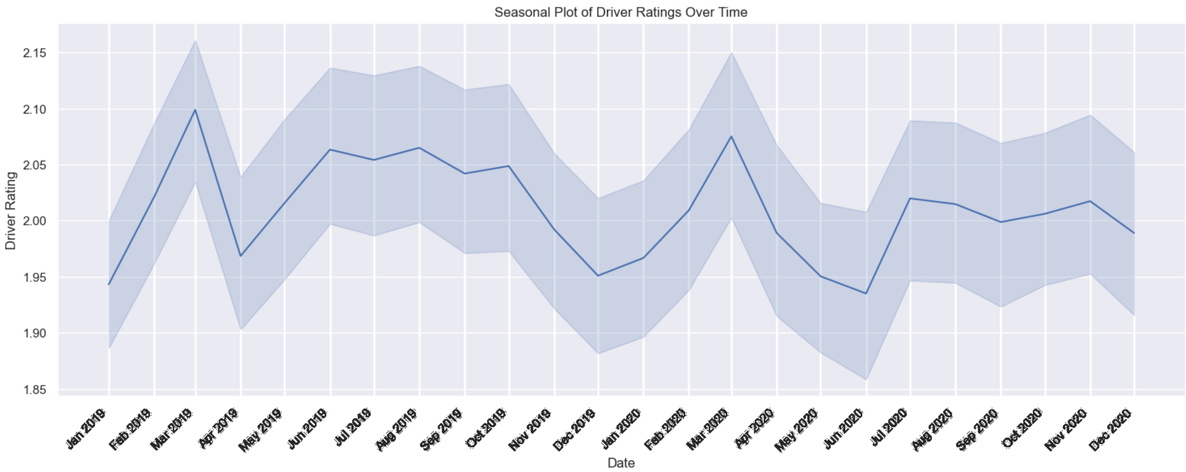


In [34]:
```
# 9. Will the driver's performance be affected by the City they operate in? (Yes/No
q9 = data[["City", "Quarterly Rating"]]

# Create a grouped bar plot using Seaborn
plt.figure(figsize=(16, 8))
sns.countplot(x='City', hue='Quarterly Rating', data=q9, palette='viridis')

plt.xlabel('City')
plt.ylabel('Count')
plt.title('Distribution of Quarterly Ratings Across Cities')
plt.show()

# There is no specific pattern can be shown for driver performance in particular ci
# Therefore, The driver's performance is not affected by the City they operate in.
```



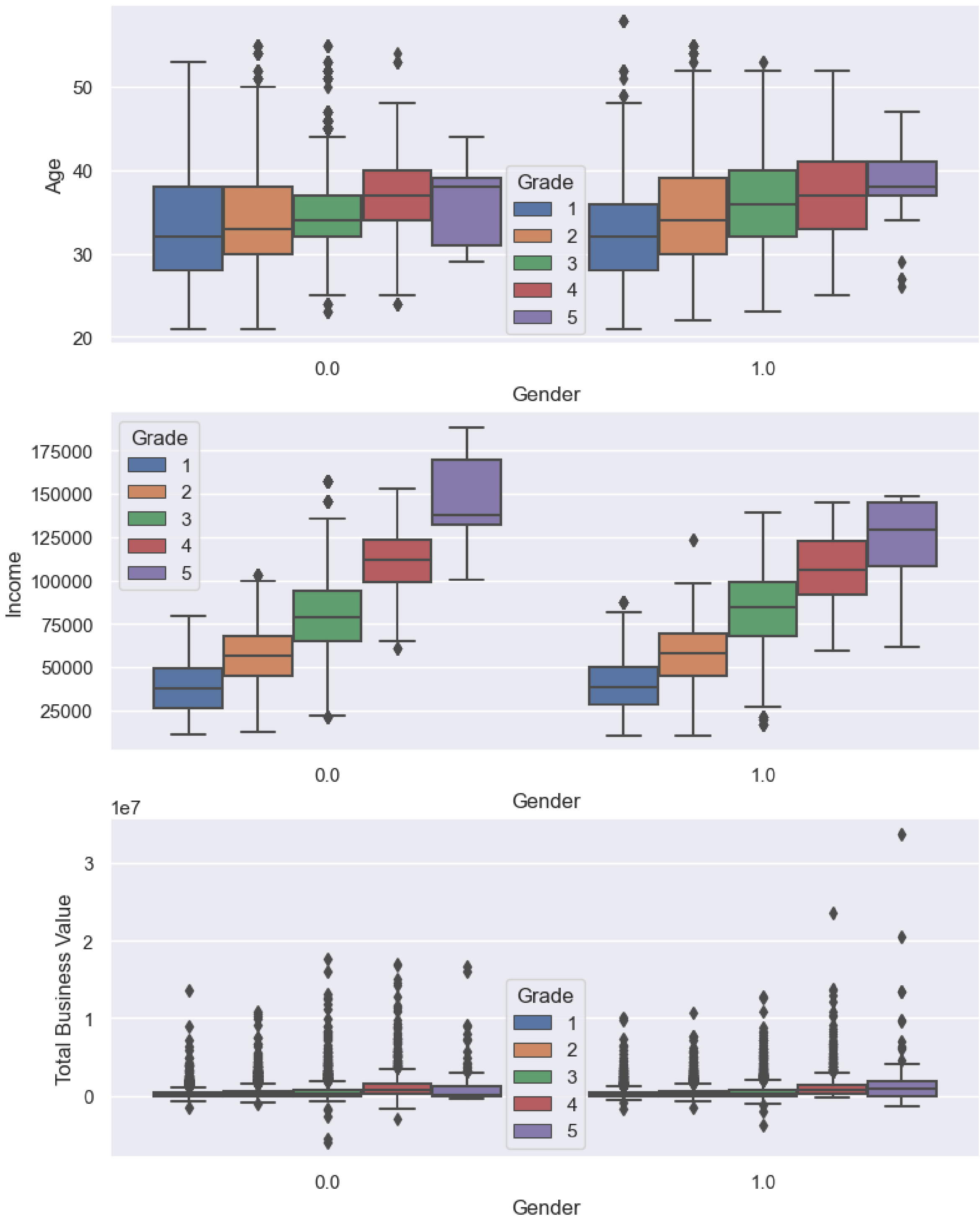
In [35]:
```
# 10.Analyze any seasonality in the driver's ratings. Do certain times of the year
# correspond to higher or lower ratings, and why might that be?
q10 = data[["MMM-YY", "Quarterly Rating"]]
```

```
# Create a seasonal plot using Seaborn
plt.figure(figsize=(18, 6))
sns.lineplot(data=q10, x="MMM-YY", y='Quarterly Rating')
plt.xticks(q10['MMM-YY'], [date.strftime('%b %Y') for date in q10['MMM-YY']], rotat

plt.xlabel('Date')
plt.ylabel('Driver Rating')
plt.title('Seasonal Plot of Driver Ratings Over Time')
plt.show()

# March 2019 showing the highest rating followed by March 2020
# June 2020 showing the lowest rating followed by Dec 2019 and Apr 2019
# There is no particular pattern for lowest rating but highest rating was seen in m
```

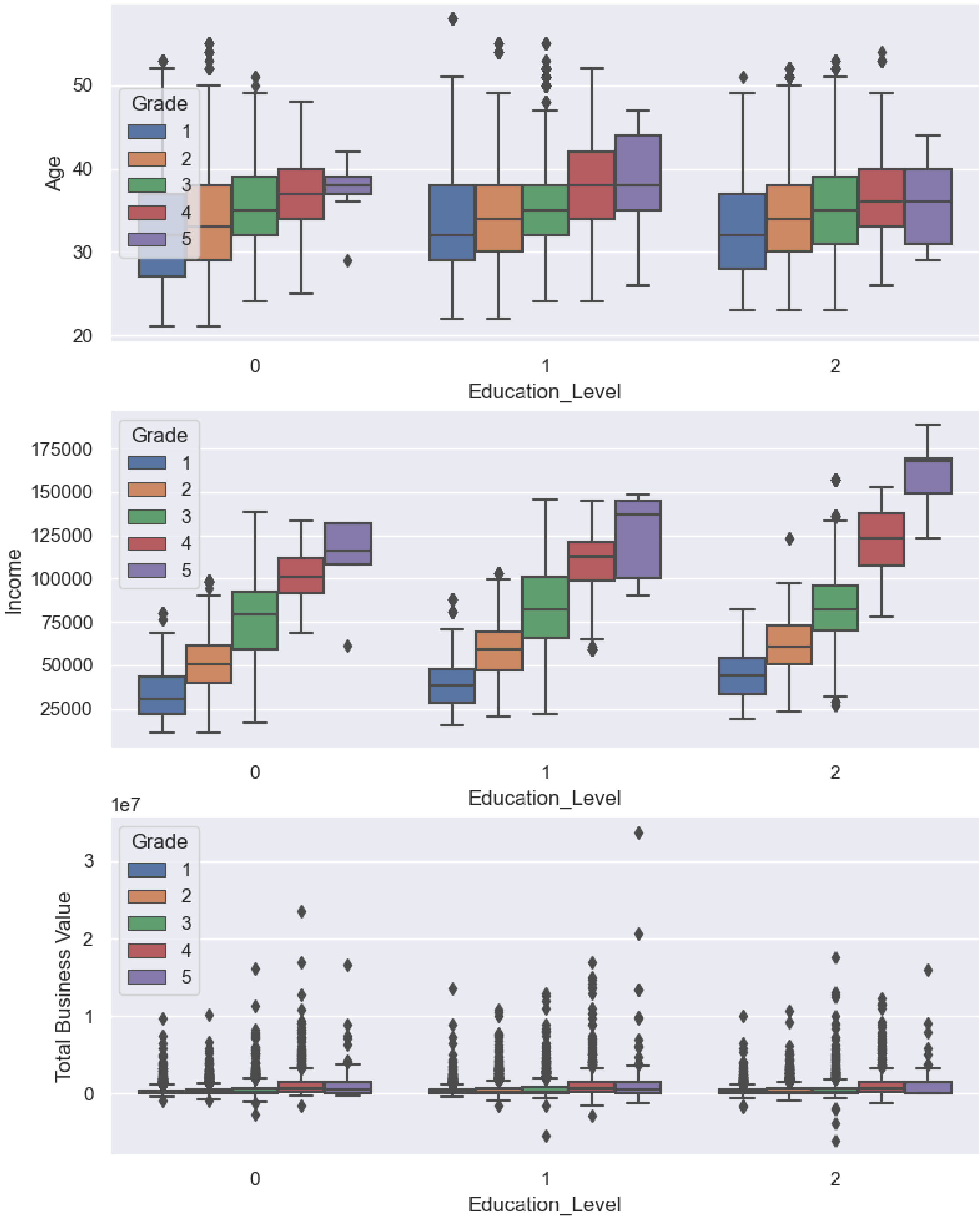


Multivariate Analysis

In [36]:
```
# Check variation of each numerical features with repect to multi categorical featu
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(9, 12), sharey='row')
for i, num_col in enumerate(num_cols):
    sns.boxplot(x="Gender", y=num_col, hue = "Grade" ,data=data, ax = axes[i])
plt.show()
```

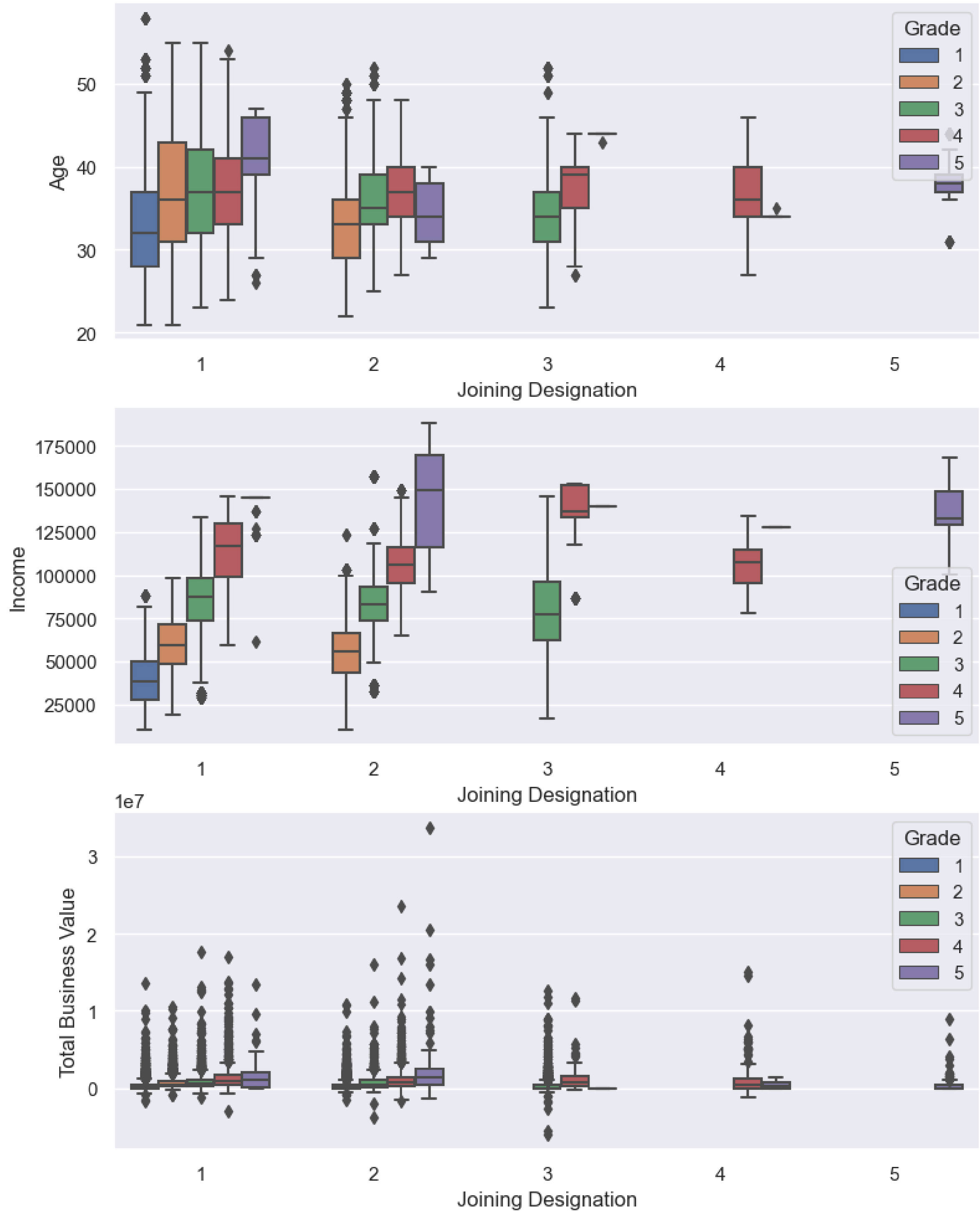


In [37]:
```
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(9, 12), sharey='row')
for i, num_col in enumerate(num_cols):
    sns.boxplot(x="Education_Level", y=num_col, hue = "Grade" ,data=data, ax = axes
plt.show()
```



In [38]:

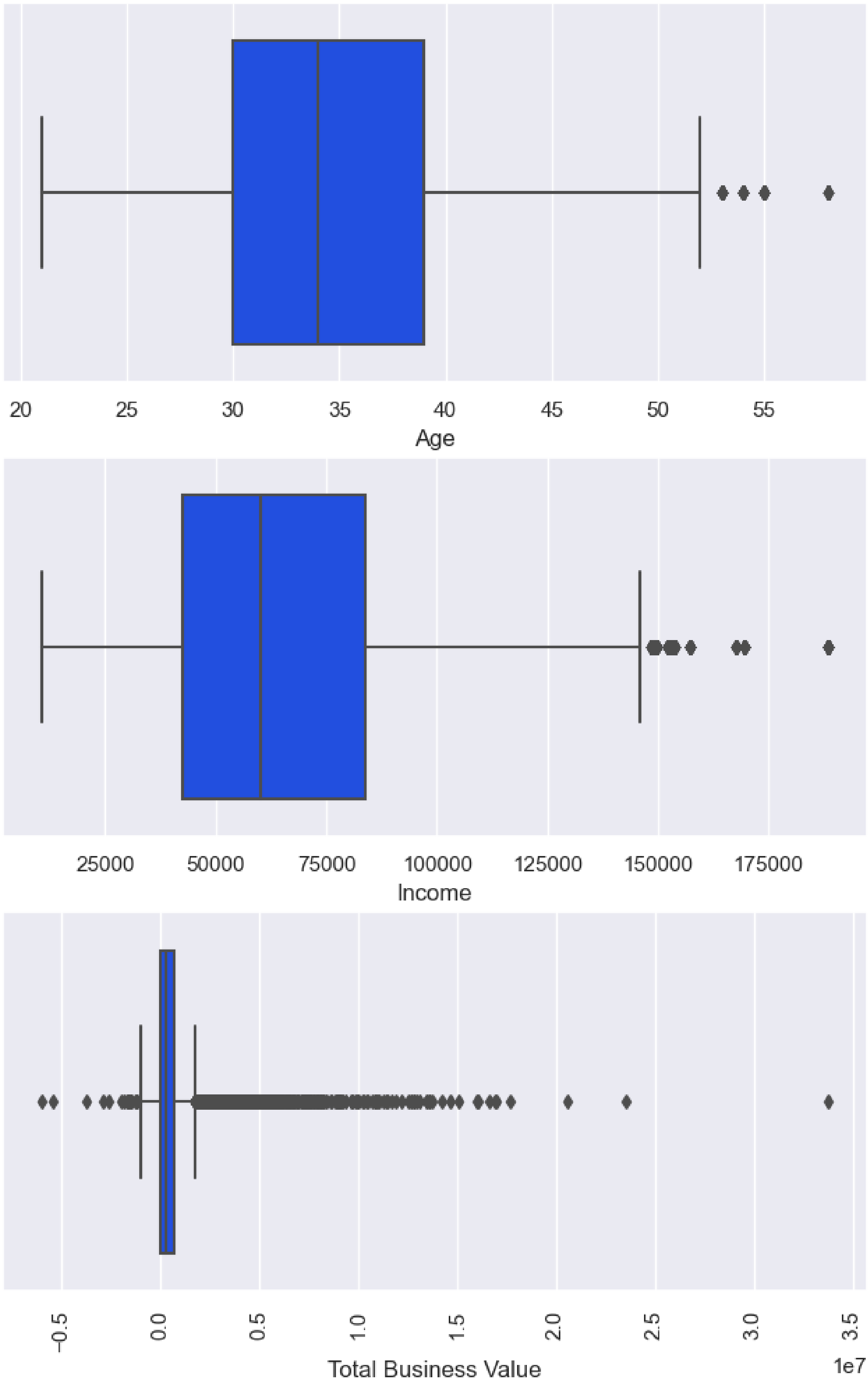
```
fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(9, 12), sharey='row')
for i, num_col in enumerate(num_cols):
    sns.boxplot(x="Joining Designation", y=num_col, hue = "Grade" ,data=data, ax =
plt.show()
```



Data Preprocessing

Check for Outliers

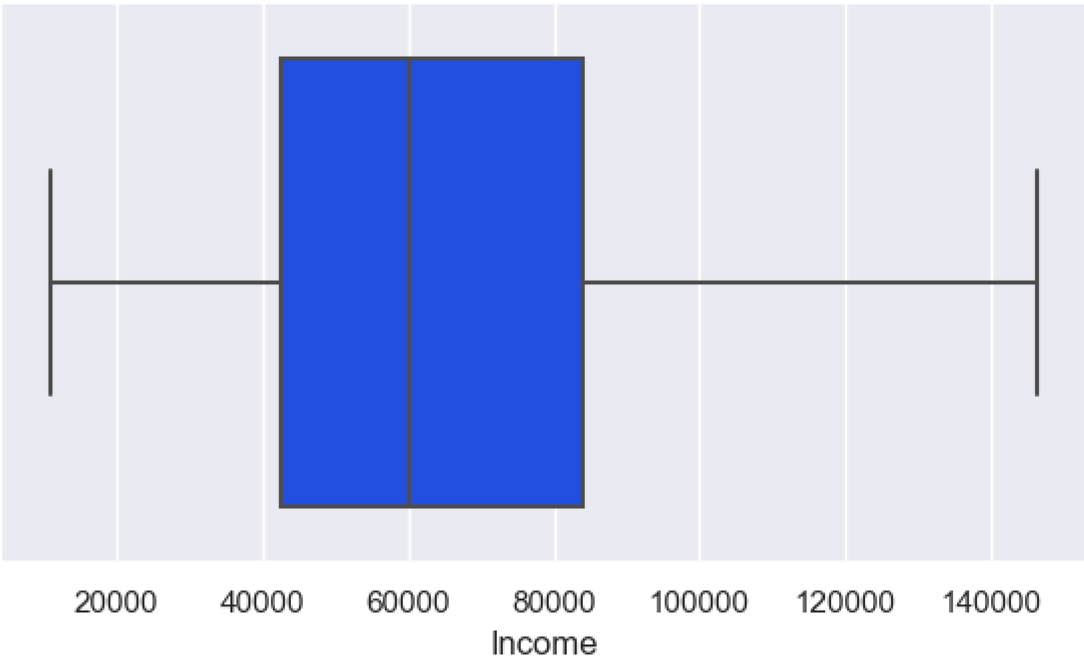
In [39]:
```
num_cols = ["Age", "Income", "Total Business Value"]
fig, axis = plt.subplots(nrows=3, ncols=1, figsize=(8, 12))
index = 0
for row in range(3):
    sns.boxplot(data[num_cols[index]], ax=axis[row], palette="bright")
    index += 1
plt.xticks(rotation=90)
plt.show()

# Features like "Income" and "Total Business Value" has many outliers. They need to
```

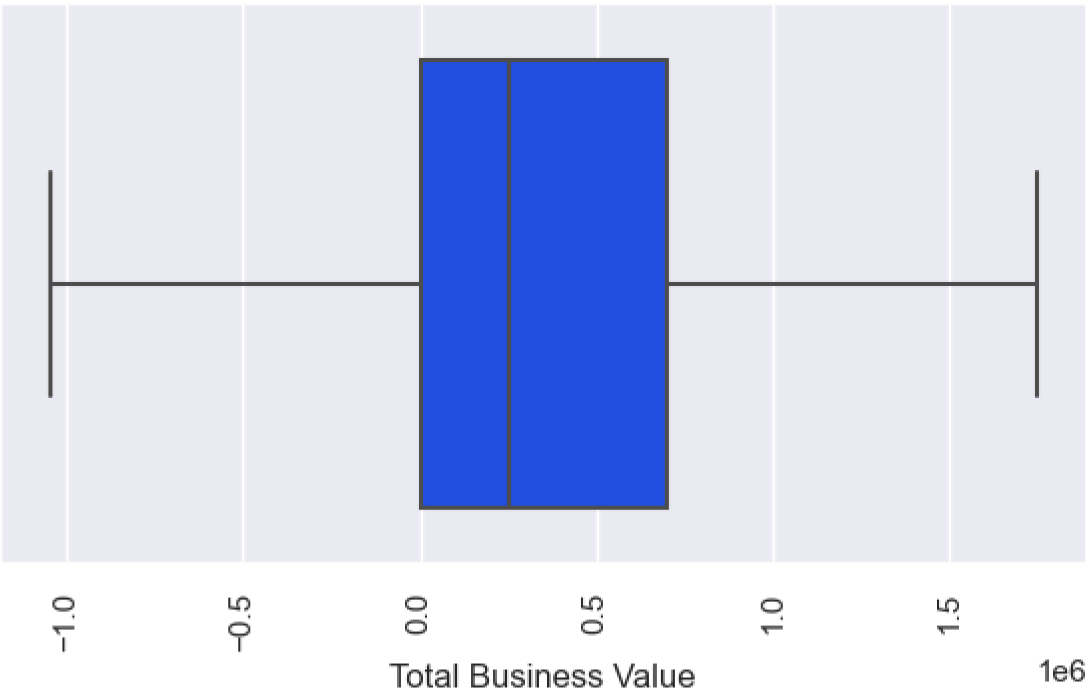


Treatment of Outliers: Quantile based flooring and capping

In [40]:
```
num_cols_1 = num_cols[1:]
fig, axis = plt.subplots(nrows=2, ncols=1, figsize=(7, 8))
```

```
index = 0
for row in range(2):
    q1 = np.percentile(data[num_cols_1[index]], 25)
    q3 = np.percentile(data[num_cols_1[index]], 75)
    IQR = q3-q1
    lower_bound = q1-(1.5*IQR)
    upper_bound = q3+(1.5*IQR)
    data[num_cols_1[index]] = np.where(data[num_cols_1[index]] < lower_bound, lower
    data[num_cols_1[index]] = np.where(data[num_cols_1[index]] > upper_bound, upper
    sns.boxplot(data[num_cols_1[index]], ax=axis[row], palette="bright")
    index += 1
plt.xticks(rotation=90)
plt.show()
```





Prepare data for KNN Imputation

In [41]:

```
# Check numerical features
# K-Nearest Neighbors (KNN) imputation is a technique used to fill missing values i
# based on the values of its nearest neighbors.
```

```
data[num_cols].isnull().sum()

# Age feature requires imputation
```

Out[41]:
```
Age                     61
Income                   0
Total Business Value     0
dtype: int64
```

In [42]:
```
# Clean Gender feature
data["Gender"].isnull().sum()

# Gender feature needs imputation
```

Out[42]:
```
52
```

In [43]:
```
df1 = data[["Age"]]
```

In [44]:
```
from sklearn.impute import KNNImputer

imputer = KNNImputer(n_neighbors=3)
imputed = imputer.fit_transform(df1)
df_imputed = pd.DataFrame(imputed, columns=df1.columns)
```

In [45]:
```
data["Age"] = df_imputed["Age"]
```

In [46]:
```
df2 = data[["Gender"]]
```

In [47]:
```
imputer = KNNImputer(n_neighbors=3)
imputed_1 = imputer.fit_transform(df2)
df_imputed_1 = pd.DataFrame(imputed_1, columns=df2.columns)
```

In [48]:
```
data["Gender"] = df_imputed_1["Gender"]
```

Aggregate data in order to remove multiple occurrences of same driver data

In [49]:
```
unique_driver_data = pd.DataFrame(data["Driver_ID"].unique(), columns=['Driver_ID']
```

Feature Engineering

In [50]:
```
data['Quarterly Rating'] = data['Quarterly Rating'].astype('int64')
data['Grade'] = data['Grade'].astype('int64')
```

Create a column which tells whether the quarterly rating has increased for that driver - for those whose quarterly rating has increased we assign the value 1

In [51]:
```
quat_rating = pd.DataFrame((data.groupby(['Driver_ID'],as_index=False).agg({'Quarte
           ).loc[:,'Quarterly Rating']-data.groupby(['Driver_ID'], as_index=False).
    {'Quarterly Rating':'last'}).loc[:,'Quarterly Rating']).apply(lambda x: 1 if x
```

Target variable creation: Create a column called target which tells whether the driver has left the company- driver whose last working day is present will have the value 1

In [52]:
```
data['LastWorkingDate'] = pd.to_datetime(data['LastWorkingDate'])
d1 = data.groupby(['Driver_ID'],as_index=False).agg({'LastWorkingDate':'max'}).fill
Target = pd.DataFrame(d1.LastWorkingDate.apply(lambda x: 1 if x != 0 else 0))
```

Create a column which tells whether the monthly income has increased for that driver - for those whose monthly income has increased we assign the value 1

In [53]:
```
inc_income = pd.DataFrame((data.groupby(['Driver_ID'],as_index=False).agg({'Income'
            ).loc[:,'Income']-data.groupby(['Driver_ID'], as_index=False).agg(
    {'Income':'last'}).loc[:,'Income']).apply(lambda x: 1 if x < 0 else 0))
```

Create a column which tells whether the grade has changed for that driver - for those whose grade has changed we assign the value 1

In [54]:
```
grade_change = pd.DataFrame((data.groupby(['Driver_ID'],as_index=False).agg({'Grade
            ).loc[:,'Grade']-data.groupby(['Driver_ID'], as_index=False).agg(
     {'Grade':'last'}).loc[:,'Grade']).apply(lambda x: 1 if x !=0 else 0))
```

Group all the features as new dataset and merge with original datset

In [55]:
```
data_grouped = pd.concat((unique_driver_data,quat_rating,inc_income,grade_change,Ta
data_grouped.rename(columns = {'Quarterly Rating':'quat_rating',
                    'Income':'inc_income',
                    'Grade':'grade_change',
                    'LastWorkingDate':'Target' }, inplace = True)
```

In [56]:
```
data_grouped.head()
```

Out[56]:

	Driver_ID	quat_rating	inc_income	grade_change	Target
0	1	0	0	0	1
1	2	0	0	0	0
2	4	0	0	0	1
3	5	0	0	0	1
4	6	1	0	0	0

Before merging, clean all the features

In [57]:
```
data[['Age','Gender','Education_Level','Joining Designation']
    ] = data[['Age','Gender','Education_Level','Joining Designation']].astype('int6
```

In [58]:
```
data_temp = data.groupby(['Driver_ID'], as_index=False).agg({
    'Age':'max', 'Gender':'max','Education_Level':'last',
    'Income':'max', 'Joining Designation':'last',
    'Grade':'max', 'Total Business Value':'sum',
    })
```

In [59]:
```
data_temp.head()
```

Out[59]:

	Driver_ID	Age	Gender	Education_Level	Income	Joining Designation	Grade	Total Business Value
0	1	28	0	2	57387.0	1	1	1083770.0
1	2	31	0	2	67016.0	2	2	0.0
2	4	43	0	2	65603.0	2	2	350000.0
3	5	29	0	0	46368.0	1	1	120360.0
4	6	31	1	1	78728.0	3	3	1265000.0

In [60]:

```
data.head()
# Need to analyse features "MMM-YY", "City", "Dateofjoining", "LastWorkignDate", "Q
```

Out[60]:

	MMM-YY	Driver_ID	Age	Gender	City	Education_Level	Income	Dateofjoining	LastWorkingDat
0	2019-01-01	1	28	0	C23	2	57387.0	2018-12-24	Na
1	2019-02-01	1	28	0	C23	2	57387.0	2018-12-24	Na
2	2019-03-01	1	28	0	C23	2	57387.0	2018-12-24	2019-03-1
3	2020-11-01	2	31	0	C7	2	67016.0	2020-11-06	Na
4	2020-12-01	2	31	0	C7	2	67016.0	2020-11-06	Na

So, till here, we have data_grouped having new added features grouped on driver id, we have data_temp having aggregated values of features grouped on driver id and Data : original dataset having those features which still need to be processed.

One-Hot Encoding for "Quaterly rating" feature

In [61]:

```
# Check duplicates for having same Driver ID with same Quaterly Rating
data[['Driver_ID','Quarterly Rating']].duplicated().sum()
```

Out[61]:

```
15081
```

In [62]:

```
# Delete all duplicates values
quat_rat = data[['Driver_ID','Quarterly Rating']].drop_duplicates()
```

In [63]:

```
quat_rat.duplicated().sum()
```

Out[63]:

```
0
```

In [64]:

```
quat_rat = pd.get_dummies(quat_rat, columns=['Quarterly Rating'])
quat_rat
# same IDs now divided into four records, Group them and extract value for each qua
```

Out[64]:

	Driver_ID	Quarterly Rating_1	Quarterly Rating_2	Quarterly Rating_3	Quarterly Rating_4
0	1	0	1	0	0
3	2	1	0	0	0
5	4	1	0	0	0
10	5	1	0	0	0
13	6	1	0	0	0
...	...	...	...	...	...
19091	2787	0	1	0	0
19094	2787	1	0	0	0
19097	2788	1	0	0	0
19098	2788	0	0	1	0
19101	2788	0	1	0	0

4023 rows × 5 columns

In [65]:
```
quat_rat = quat_rat.groupby(['Driver_ID'], as_index=False).agg({
    'Quarterly Rating_1':'sum','Quarterly Rating_2':'sum',
    'Quarterly Rating_3':'sum','Quarterly Rating_4':'sum'})
```

In [66]:
```
quat_rat
```

Out[66]:

	Driver_ID	Quarterly Rating_1	Quarterly Rating_2	Quarterly Rating_3	Quarterly Rating_4
0	1	0	1	0	0
1	2	1	0	0	0
2	4	1	0	0	0
3	5	1	0	0	0
4	6	1	1	0	0
...	...	...	...	...	...
2376	2784	1	0	1	1
2377	2785	1	0	0	0
2378	2786	1	1	0	0
2379	2787	1	1	0	0
2380	2788	1	1	1	0

2381 rows × 5 columns

Storing unique Driver IDs in an empty dataframe and then bring all the features at same level

In [67]:
```
data_1 = pd.merge(data_grouped, data_temp, on='Driver_ID', how='inner')
data_1.head()
```

Out[67]:

	Driver_ID	quat_rating	inc_income	grade_change	Target	Age	Gender	Education_Level	Incon
0	1	0	0	0	1	28	0	2	5738
1	2	0	0	0	0	31	0	2	6701
2	4	0	0	0	1	43	0	2	6560
3	5	0	0	0	1	29	0	0	4636
4	6	1	0	0	0	31	1	1	7872

In [68]:
```
data_1.shape
```

Out[68]:
```
(2381, 12)
```

In [69]:
```
data_2 = pd.merge(data_1, quat_rat, on='Driver_ID', how='inner')
data_2.head()
```

Out[69]:

	Driver_ID	quat_rating	inc_income	grade_change	Target	Age	Gender	Education_Level	Incon
0	1	0	0	0	1	28	0	2	5738
1	2	0	0	0	0	31	0	2	6701
2	4	0	0	0	1	43	0	2	6560
3	5	0	0	0	1	29	0	0	4636
4	6	1	0	0	0	31	1	1	7872

In [70]:
```
data_2.shape
```

Out[70]:
```
(2381, 16)
```

Target Encoding for "City" column -part 1

In [71]:
```
# City column have 29 distinct values, One-hot encoding will create extra 28 featur
# Therefore, target encoding is better.
city_data = data.groupby(['Driver_ID'], as_index=False).agg({'City':'max'})

# Target feature required for target encoding for city column.
# Therefore, merge all the newly formed datasets on basis of driver ID.
```

In [72]:
```
city_data.shape
```

Out[72]:
```
(2381, 2)
```

In [73]:
```
data_3 = pd.merge(data_2, city_data, on='Driver_ID', how='inner')
data_3.head()
```

Out[73]:

	Driver_ID	quat_rating	inc_income	grade_change	Target	Age	Gender	Education_Level	Incon
0	1	0	0	0	1	28	0	2	57387
1	2	0	0	0	0	31	0	2	67016
2	4	0	0	0	1	43	0	2	65603
3	5	0	0	0	1	29	0	0	46368
4	6	1	0	0	0	31	1	1	78728

In [74]:
```
data_3.shape
```

Out[74]: (2381, 17)

In [75]:
```
city_data_1 = data_3.groupby('City')['Target'].mean().to_dict()
```

In [76]:
```
data_3['City'] = data_3['City'].map(city_data_1)
```

In [77]:
```
data_3.head()
```

Out[77]:

	Driver_ID	quat_rating	inc_income	grade_change	Target	Age	Gender	Education_Level	Incon
0	1	0	0	0	1	28	0	2	57387
1	2	0	0	0	0	31	0	2	67016
2	4	0	0	0	1	43	0	2	65603
3	5	0	0	0	1	29	0	0	46368
4	6	1	0	0	0	31	1	1	78728

Our data is now all numerical. Now, need to analyse:

1. Statistical summary of the derived dataset
2. Check correlation among independent variables and how they interact with each other
3. Check for Class Imbalance. If yes, Treatment for the same
4. Standardization of data

In [78]:
```
data_3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2381 entries, 0 to 2380
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Driver_ID             2381 non-null   int64  
 1   quat_rating           2381 non-null   int64  
 2   inc_income            2381 non-null   int64  
 3   grade_change          2381 non-null   int64  
 4   Target                2381 non-null   int64  
 5   Age                   2381 non-null   int64  
 6   Gender                2381 non-null   int64  
 7   Education_Level       2381 non-null   int64  
 8   Income                2381 non-null   float64
 9   Joining Designation   2381 non-null   int64  
 10  Grade                 2381 non-null   int64  
 11  Total Business Value  2381 non-null   float64
 12  Quarterly Rating_1    2381 non-null   uint8  
 13  Quarterly Rating_2    2381 non-null   uint8  
 14  Quarterly Rating_3    2381 non-null   uint8  
 15  Quarterly Rating_4    2381 non-null   uint8  
 16  City                  2381 non-null   float64
dtypes: float64(3), int64(10), uint8(4)
memory usage: 269.7 KB
```

Statistical summary of derived dataset

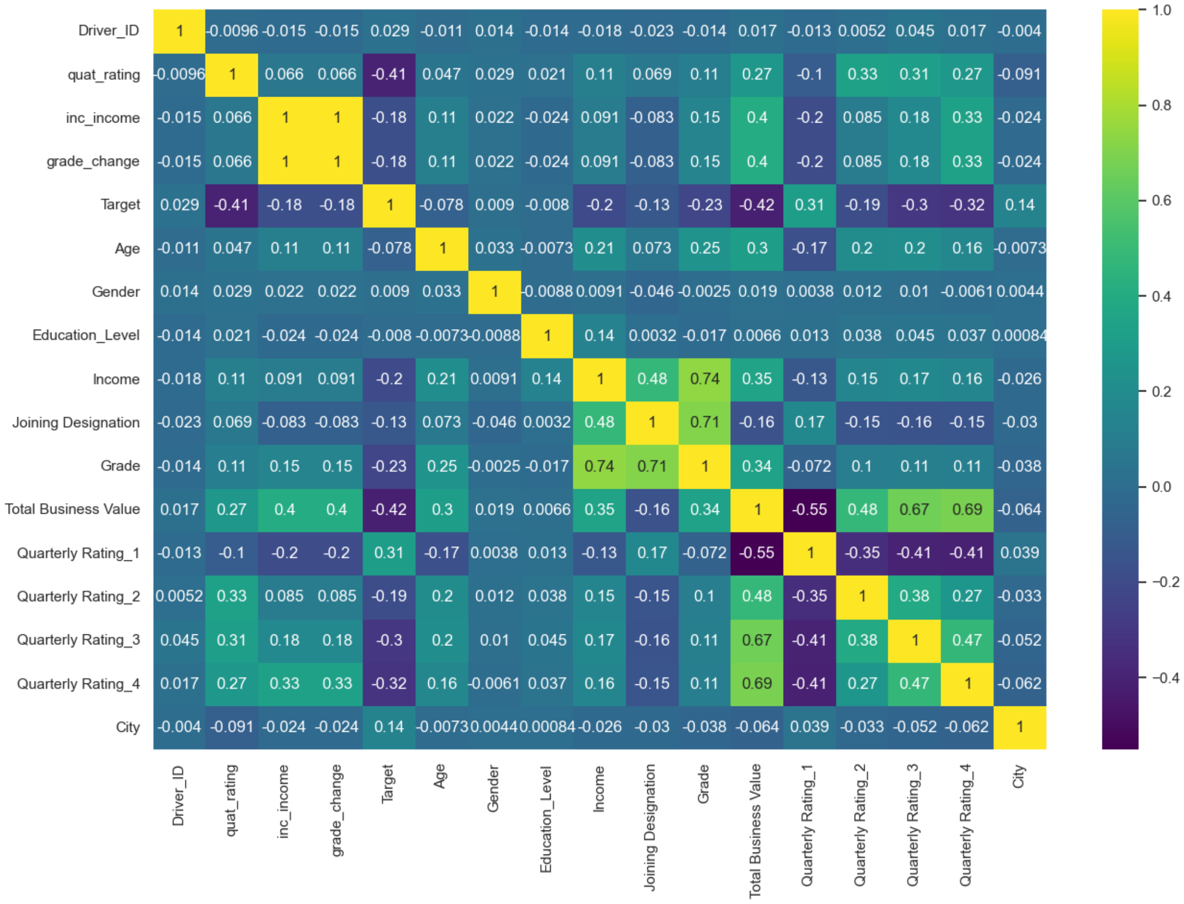
In [79]:
```
data_3.describe()
```

Out[79]:

	Driver_ID	quat_rating	inc_income	grade_change	Target	Age	Gende
count	2381.000000	2381.000000	2381.000000	2381.000000	2381.000000	2381.000000	2381.00000
mean	1397.559009	0.150357	0.018060	0.018060	0.678706	33.790004	0.41033
std	806.161628	0.357496	0.133195	0.133195	0.467071	5.907800	0.49199
min	1.000000	0.000000	0.000000	0.000000	0.000000	21.000000	0.00000
25%	695.000000	0.000000	0.000000	0.000000	0.000000	30.000000	0.00000
50%	1400.000000	0.000000	0.000000	0.000000	1.000000	33.000000	0.00000
75%	2100.000000	0.000000	0.000000	0.000000	1.000000	37.000000	1.00000
max	2788.000000	1.000000	1.000000	1.000000	1.000000	58.000000	1.00000

Check correlation among independent variables and how they interact with each other

In [80]:
```
plt.figure(figsize = (15,10))
corr = data_3.corr()
sns.heatmap(data = corr,annot = True,cmap='viridis' )
plt.show()

# Total Bussiness Value has good correlation with Quaterly Rating_3 and Quaterly Ra
# this implies that better the rating, more bussiness value generated by ola driver
# Grade has also high value of correlation with Income and Joining Designation
```



In [81]:
```
# 9. Will the driver's performance be affected by the City they operate in? (Yes/No
q_9 = data_3[["City", "Quarterly Rating_1","Quarterly Rating_2","Quarterly Rating_3
q_9.corr()

# There is very less correlation (ranging from -0.406516 to 0.474323) between City
# Therefore, driver's performance is not affected by the City they opertae in.
```

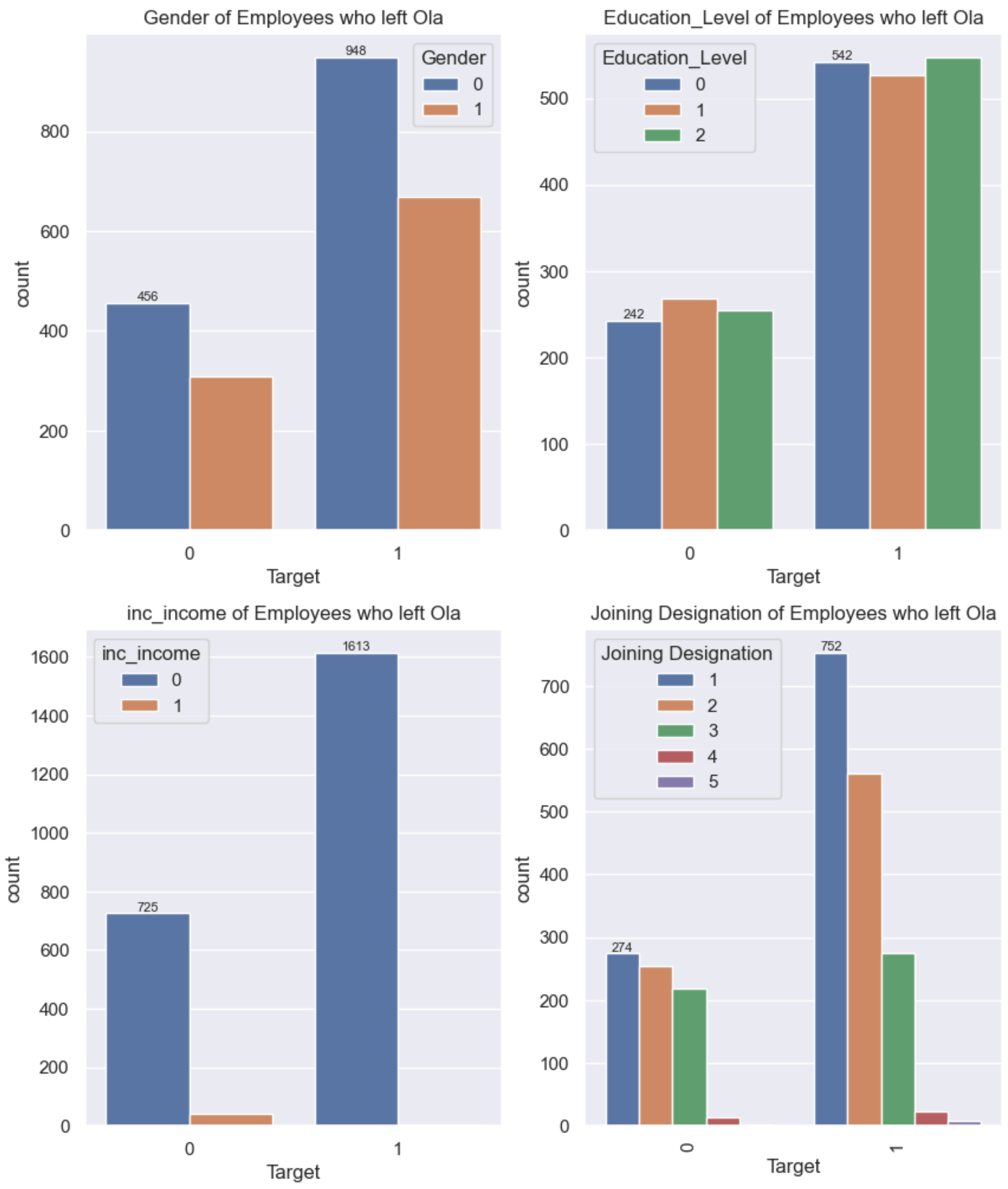
Out[81]:

	City	Quarterly Rating_1	Quarterly Rating_2	Quarterly Rating_3	Quarterly Rating_4
City	1.000000	0.038620	-0.032890	-0.052418	-0.062346
Quarterly Rating_1	0.038620	1.000000	-0.351463	-0.405251	-0.406516
Quarterly Rating_2	-0.032890	-0.351463	1.000000	0.383681	0.265250
Quarterly Rating_3	-0.052418	-0.405251	0.383681	1.000000	0.474323
Quarterly Rating_4	-0.062346	-0.406516	0.265250	0.474323	1.000000

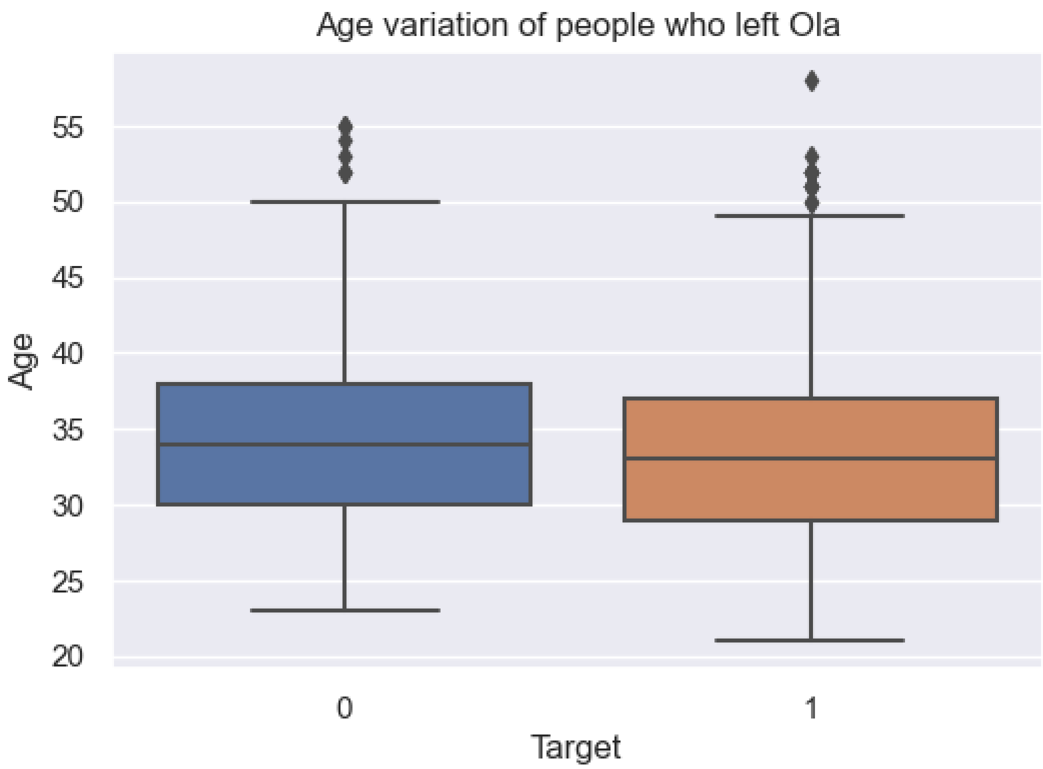
Exploratory analysis of derived dataset

In [82]:
```
# Variation of categorical features with repect to Target Variable
cols = ["Gender", "Education_Level", "inc_income","Joining Designation"]
fig, axis = plt.subplots(nrows=2, ncols=2, figsize=(10, 12))
index = 0

for row in range(2):
    for col_index in range(2):  # Use a different variable name for the inner loop
        sns.set(rc={'figure.figsize':(5, 4)})
```

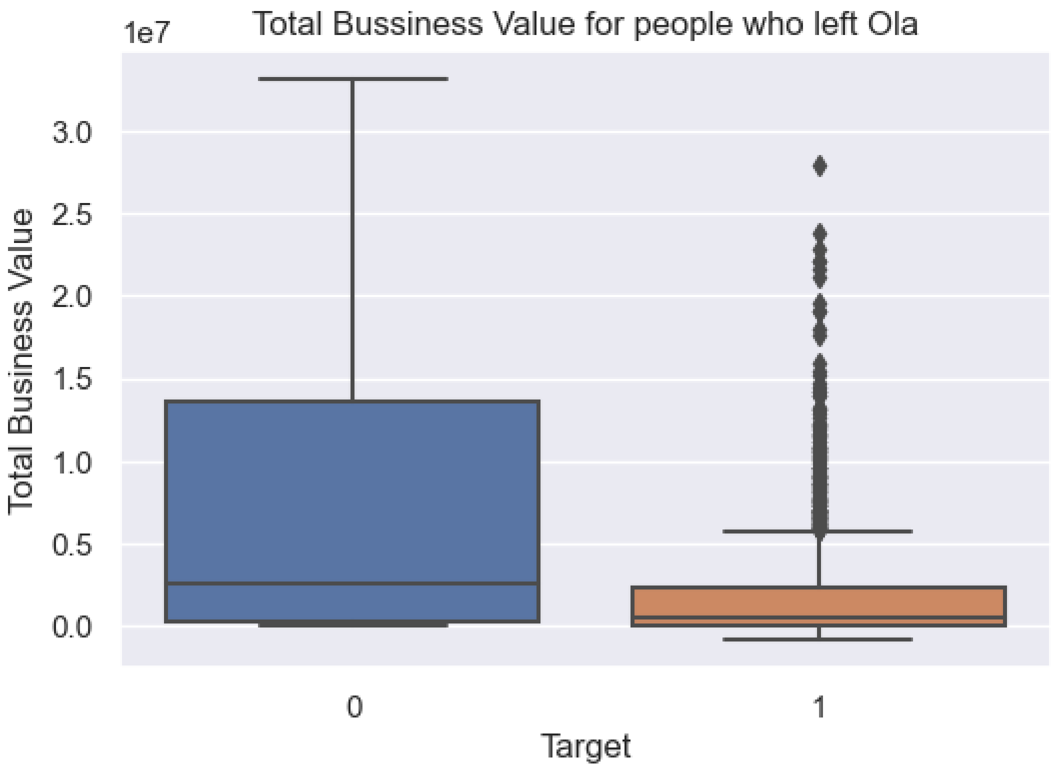
```
        ax = sns.countplot(data=data_3, x="Target", hue=data_3[cols[index]], ax=axi
        axis[row, col_index].set_title(f'{cols[index]} of Employees who left Ola')
        ax.bar_label(container=ax.containers[0], fontsize=8)
        index += 1

plt.xticks(rotation=90)
plt.show()
```



In [83]:
```
plt.figure(figsize = (6,4))
sns.boxplot(data=data_3, x="Target", y="Age")
plt.title('Age variation of people who left Ola')
plt.show()
```



In [84]:
```
plt.figure(figsize = (6,4))
sns.boxplot(data=data_3, x="Target", y="Total Business Value")
plt.title('Total Bussiness Value for people who left Ola')
plt.show()
```



Class Imbalanced

In [85]:
```
data_3.head(2)
```

Out[85]:

	Driver_ID	quat_rating	inc_income	grade_change	Target	Age	Gender	Education_Level	Incon
0	1	0	0	0	1	28	0	2	57387
1	2	0	0	0	0	31	0	2	67016

In [86]:

```
cat_columns = ['quat_rating', 'inc_income', 'grade_change', 'Target',
       'Gender', 'Education_Level', 'Quarterly Rating_1',
       'Quarterly Rating_2', 'Quarterly Rating_3', 'Quarterly Rating_4'
       ]
```

In [87]:

```
# Check for class imbalance
for col in cat_columns:
    per = data_3[col].value_counts()
    print("\033[1mcolumn name :",col,"\033[0m")
    for i in range(len(per)):
        print('percent availability of data for',i,'is',((per[i]/len(data_3[col]))*
    print("---------------------------------------------------------")
```

```
column name : quat_rating
percent availability of data for 0 is 84.96 %
percent availability of data for 1 is 15.04 %
---------------------------------------------------------
column name : inc_income
percent availability of data for 0 is 98.19 %
percent availability of data for 1 is 1.81 %
---------------------------------------------------------
column name : grade_change
percent availability of data for 0 is 98.19 %
percent availability of data for 1 is 1.81 %
---------------------------------------------------------
column name : Target
percent availability of data for 0 is 32.13 %
percent availability of data for 1 is 67.87 %
---------------------------------------------------------
column name : Gender
percent availability of data for 0 is 58.97 %
percent availability of data for 1 is 41.03 %
---------------------------------------------------------
column name : Education_Level
percent availability of data for 0 is 32.93 %
percent availability of data for 1 is 33.39 %
percent availability of data for 2 is 33.68 %
---------------------------------------------------------
column name : Quarterly Rating_1
percent availability of data for 0 is 12.43 %
percent availability of data for 1 is 87.57 %
---------------------------------------------------------
column name : Quarterly Rating_2
percent availability of data for 0 is 58.76 %
percent availability of data for 1 is 41.24 %
---------------------------------------------------------
column name : Quarterly Rating_3
percent availability of data for 0 is 74.13 %
percent availability of data for 1 is 25.87 %
---------------------------------------------------------
column name : Quarterly Rating_4
percent availability of data for 0 is 85.72 %
percent availability of data for 1 is 14.28 %
---------------------------------------------------------
```

Observe that most of the features showing imbalanced data. Therefore, this class imbalancve needs to be treated.

1. data for "Gender", "Education_Level" is balanced.
2. data for "inc_income",grade_change, and quat_rating are highly imbalanced
3. data of target variable also needs imbalance treatment

Treatment for Class Imbalance

In [88]:
```
data_3.head()
```

Out[88]:

	Driver_ID	quat_rating	inc_income	grade_change	Target	Age	Gender	Education_Level	Incon
0	1	0	0	0	1	28	0	2	57387
1	2	0	0	0	0	31	0	2	67016
2	4	0	0	0	1	43	0	2	65603
3	5	0	0	0	1	29	0	0	46368
4	6	1	0	0	0	31	1	1	78728

In [89]:

```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# 'features' should contain the feature columns, and 'target' should contain the ta
features = data_3.drop("Target", axis=1)
target = data_3["Target"]

# Split the data into training and testing sets
features_train, features_test, target_train, target_test = train_test_split(feature
                                target, test_size=0.2, random_state=42)

# Initialize SMOTE for each feature independently
smote_quat_rating = SMOTE(random_state=42)
smote_inc_income = SMOTE(random_state=42)
smote_grade_change = SMOTE(random_state=42)
smote_target = SMOTE(random_state=42)

# Apply SMOTE to each feature
features_train_resampled_quat_rating, target_train_resampled_quat_rating = smote_qu
    features_train[['quat_rating']], target_train)
features_train_resampled_inc_income, target_train_resampled_inc_income = smote_inc_
    features_train[['inc_income']], target_train)
features_train_resampled_grade_change, target_train_resampled_grade_change = smote_
    features_train[['grade_change']], target_train)
features_train_resampled_target, target_train_resampled_target = smote_target.fit_r
    ['quat_rating', 'inc_income', 'grade_change'], axis=1), target_train)

# Concatenate the resampled features
features_train_resampled = pd.concat([pd.DataFrame(features_train_resampled_quat_ra
                                      pd.DataFrame(features_train_resampled_inc_inco
                                      pd.DataFrame(features_train_resampled_grade_ch
                                      features_train_resampled_target], axis=1)
```

In [90]:

```
data_4 = pd.concat((features_train_resampled,target_train_resampled_target), axis =
```

In [91]:

```
data_4.head(2)
```

Out[91]:

	quat_rating	inc_income	grade_change	Driver_ID	Age	Gender	Education_Level	Income	Desi
0	0	0	0	534	28	0	0	87872.0	
1	0	0	0	2044	34	0	0	38619.0	

In [92]:
```python
# Check for class imbalance
for col in cat_columns:
    per = data_4[col].value_counts()
    print("\033[1mcolumn name :",col,"\033[0m")
    for i in range(len(per)):
        print('percent availability of data for',i,'is',((per[i]/len(data_4[col]))*
    print("---------------------------------------------------------")
```

```
column name : quat_rating
percent availability of data for 0 is 79.05 %
percent availability of data for 1 is 20.95 %
---------------------------------------------------------
column name : inc_income
percent availability of data for 0 is 97.44 %
percent availability of data for 1 is 2.56 %
---------------------------------------------------------
column name : grade_change
percent availability of data for 0 is 97.44 %
percent availability of data for 1 is 2.56 %
---------------------------------------------------------
column name : Target
percent availability of data for 0 is 50.0 %
percent availability of data for 1 is 50.0 %
---------------------------------------------------------
column name : Gender
percent availability of data for 0 is 65.9 %
percent availability of data for 1 is 34.1 %
---------------------------------------------------------
column name : Education_Level
percent availability of data for 0 is 34.72 %
percent availability of data for 1 is 37.66 %
percent availability of data for 2 is 27.62 %
---------------------------------------------------------
column name : Quarterly Rating_1
percent availability of data for 0 is 19.63 %
percent availability of data for 1 is 80.37 %
---------------------------------------------------------
column name : Quarterly Rating_2
percent availability of data for 0 is 57.64 %
percent availability of data for 1 is 42.36 %
---------------------------------------------------------
column name : Quarterly Rating_3
percent availability of data for 0 is 71.92 %
percent availability of data for 1 is 28.08 %
---------------------------------------------------------
column name : Quarterly Rating_4
percent availability of data for 0 is 83.05 %
percent availability of data for 1 is 16.95 %
---------------------------------------------------------
```

1. Target variable is now balanced.
2. Observe that variables like quat_rating, inc_income, and grade_change are little bit better balanced than previous dataset.

Standardization of Training Data

Standardization needs to be done on "Age", "Education_Level","Income", "Joining Designation", "Grade", and "Total Business value"

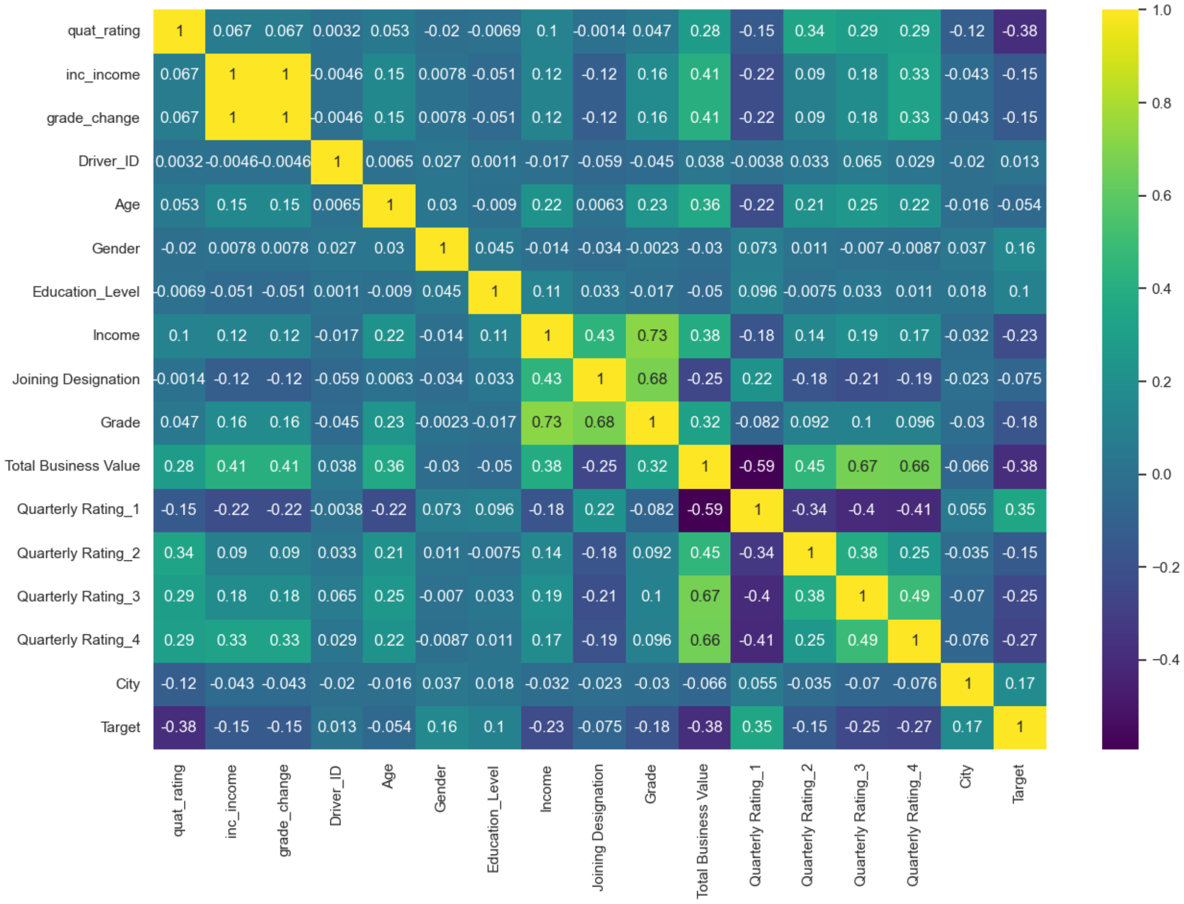
In [93]:
```python
data_4.head(2)
```

Out[93]:

	quat_rating	inc_income	grade_change	Driver_ID	Age	Gender	Education_Level	Income	Desi
0	0	0	0	534	28	0	0	87872.0	
1	0	0	0	2044	34	0	0	38619.0	

In [94]:
```python
std_cols = ['Age','Education_Level','Income','Joining Designation','Grade','Total B
for col in std_cols:
    data_4[col] = (data_4[col] - np.mean(data_4[col]))/np.std(data_4[col])
```

In [95]:
```python
# Before proceedign for Modelling, check the correlation between all features, so t
plt.figure(figsize = (15,10))
corr = data_4.corr()
sns.heatmap(data = corr,annot = True,cmap='viridis' )
plt.show()
```



In [96]:
```python
# 'inc_income' and 'grade_change' having perfect correlation. so delete one feature
#  'Grade' is showing correlation with 'Income' and 'Joining Designation' but there
# 'Income' and 'Joining Designation', Therefore, deleting 'Grade' feature will be h
# Driver ID feature is not relevant in ML modeling. Hence, delete this feature.

data_4 = data_4.drop(["grade_change", "Grade"], axis=1)
```

In [97]:
```python
data_4 = data_4.drop("Driver_ID", axis=1)
```

In [98]:
```python
data_4.head()
```

Out[98]:

	quat_rating	inc_income	Age	Gender	Education_Level	Income	Joining Designation	Total Business Value
0	0	0	-1.038893	0	-1.181459	0.934819	1.436394	-0.650615
1	0	0	0.015819	0	-1.181459	-0.830114	0.225860	-0.583752
2	0	0	-1.214678	0	-1.181459	-0.311022	-0.984674	-0.650615
3	1	0	-0.687322	1	1.362007	-0.987246	-0.984674	-0.150169
4	0	0	2.301029	0	-1.181459	1.028417	-0.984674	1.954270

In [99]:

```
data_4.shape
```

Out[99]:

```
(2578, 14)
```

Model building

Ensemble Learning- Bagging Algorithm

Bagging (Random Forest)

In [100…

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,precision_score,

# 'features' should contain the feature columns, and 'target' should contain the ta
features = data_4.drop("Target", axis=1)
target = data_4["Target"]

# Split the data into training and testing sets
features_train, features_test, target_train, target_test = train_test_split(feature
                                                target, test_size=0.2, random_state
```

In [101…

```
# Initialize the Random Forest classifier
rf_classifier = RandomForestClassifier(random_state=42)

# Train the Random Forest model
rf_classifier.fit(features_train, target_train)

# Make predictions on the test set
target_pred_rf = rf_classifier.predict(features_test)
```

Classification Report

In [102…

```
# Evaluate the Random Forest model
print("Random Forest Classification Report:")
print(classification_report(target_test, target_pred_rf))
print("Accuracy:", accuracy_score(target_test, target_pred_rf))
print('F1 score:',f1_score(target_test, target_pred_rf))
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.87      0.85       254
           1       0.87      0.82      0.84       262

    accuracy                           0.84       516
   macro avg       0.84      0.84      0.84       516
weighted avg       0.84      0.84      0.84       516

Accuracy: 0.8430232558139535
F1 score: 0.8408644400785854
```

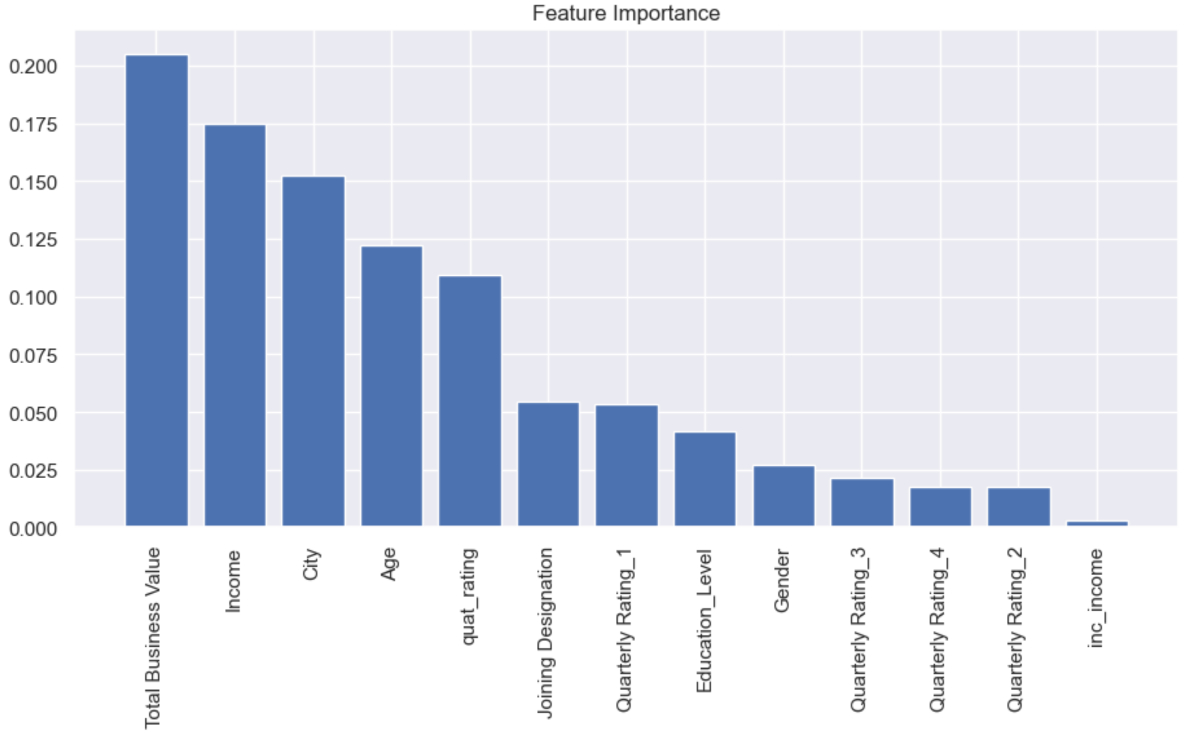
1. Precision is the ratio of correctly predicted positive observations to the total predicted positives.
   A. For class 0, precision is 0.82, meaning that out of all instances predicted as class 0, 82% were actually class 0.
   B. For class 1, precision is 0.87, indicating that 87% of instances predicted as class 1 were indeed class 1.
2. Recall (or Sensitivity) is the ratio of correctly predicted positive observations to the total actual positives.
   A. For class 0, recall is 0.87, meaning that the model correctly identified 87% of all actual class 0 instances.
   B. For class 1, recall is 0.82, indicating that the model captured 82% of all actual class 1 instances.
3. The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics.
   A. The weighted average F1-score is 0.8409, indicating overall good performance.
4. The overall accuracy of the model is 0.843, which means it correctly predicted the class labels for approximately 84.3% of the instances.

```
 The model seems to perform reasonably well with a balanced trade-off
between precision and recall.
 Both classes (0 and 1) have similar accuracy, precision, and recall,
indicating a good overall balance
 in classification.
 The F1-score is also quite high, suggesting a good compromise between
precision and recall.
```

Feature Importance

In [103…

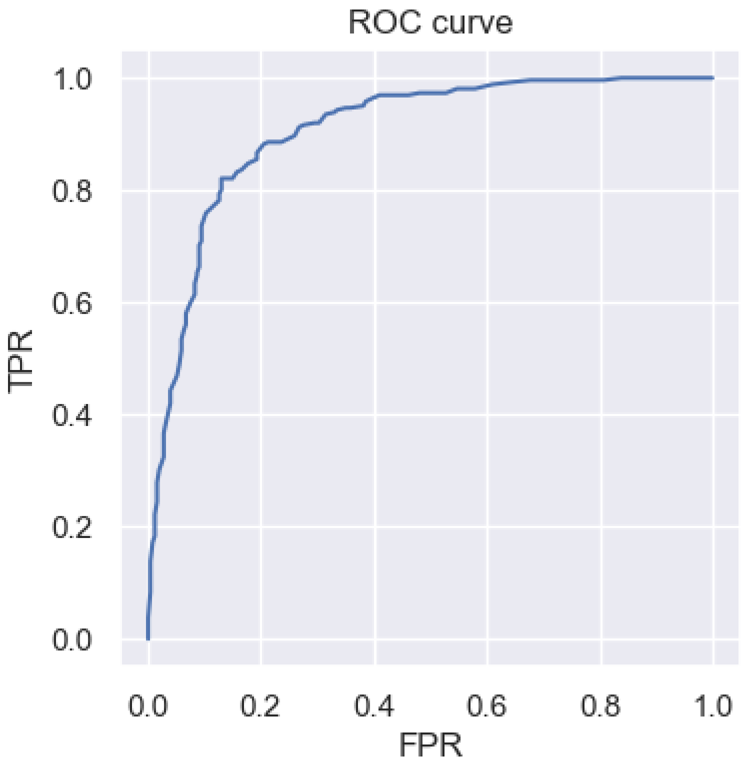
```
importance = rf_classifier.feature_importances_
indices = np.argsort(importance)[::-1] # Sort feature importances in descending ord
names = [data_4.columns[i] for i in indices] # Rearrange feature names so they matc
plt.figure(figsize=(11, 5)) # Create plot
plt.title("Feature Importance") # Create plot title
plt.bar(range(features.shape[1]), importance[indices]) # Add bars
plt.xticks(range(features.shape[1]), names, rotation=90) # Add feature names as x-a
plt.show() # Show plot
```



ROC AUC Curve

In [104…

```
from sklearn.metrics import roc_curve, roc_auc_score

pred_prob2 = rf_classifier.predict_proba(features_test)
fpr, tpr, thr = roc_curve(target_test,np.ravel(pred_prob2[:,1]))
plt.subplots(figsize=(4,4))
plt.plot(fpr,tpr)
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
print('Area under ROC AUC Curve :', roc_auc_score(target_test,np.ravel(pred_prob2[:
```



```
Area under ROC AUC Curve : 0.9055493778926489
```

```
An AUC score of 0.9055 is quite high and indicates that model has a
strong ability to discriminate
between the positive and negative classes.
A score of 1.0 would represent a perfect classifier, so a value
close to 1 suggests good performance.
```

Confusion Matrix

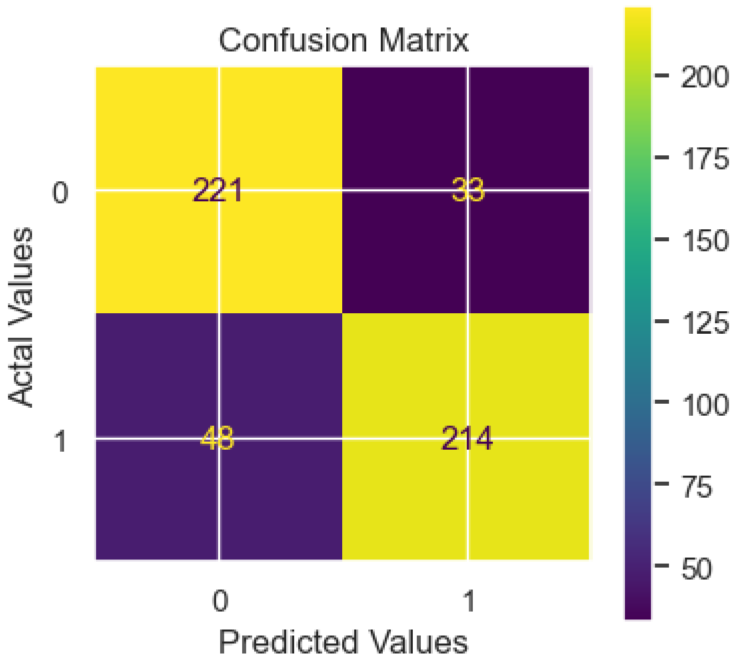
In [105…

```
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

conf_mat = confusion_matrix(target_test, rf_classifier.predict(features_test))
print(conf_mat)
```

```
[[221  33]
 [ 48 214]]
```

In [106…

```
#Plotting the confusion matrix
fig, ax = plt.subplots(figsize=(4,4))
cmp = ConfusionMatrixDisplay(conf_mat, display_labels=np.arange(2))
cmp.plot(ax=ax)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



1. True Positives (TP): 214 - Instances correctly predicted as positive (class 1).
2. True Negatives (TN): 221 - Instances correctly predicted as negative (class 0).
3. False Positives (FP): 33 - Instances predicted as positive but actually belong to the negative class.
4. False Negatives (FN): 48 - Instances predicted as negative but actually belong to the positive class.

```
  The model shows high accuracy in true positives (214) and true
negatives (221).
  The low false positives (33) and false negatives (48) suggest good
precision and recall.
  Sensitivity (0.8169) and specificity (0.8700) values reinforce the
model's ability to
  identify positive and negative instances.
```

Hyper-Parameter Tuning

Hyper tuning for MAX_DEPTH

In [107…

```
# GridSearchCV to find optimal n_estimators
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV


# specify number of folds for k-fold CV
n_folds = 5

# parameters to build the model on
parameters = {'max_depth': range(4, 20, 2)}

# instantiate the model
rf = RandomForestClassifier()


# fit tree on training data
rf = GridSearchCV(rf, parameters, cv=n_folds, scoring="accuracy",return_train_score
rf.fit(features_train, target_train)

# scores of GridSearch CV
scores = rf.cv_results_
pd.DataFrame(scores).head(2)
```

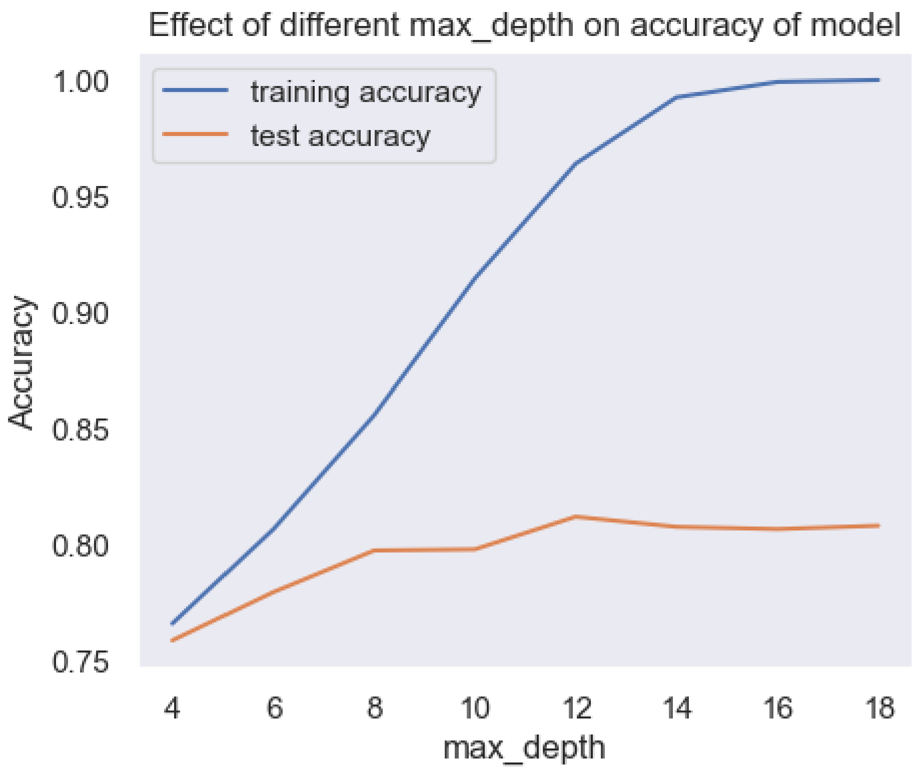
Out[107]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params
0	0.182123	0.006939	0.009157	0.005574	4	{'max_depth': 4}
1	0.211024	0.025810	0.004865	0.006342	6	{'max_depth': 6}

2 rows × 21 columns

In [108…

```
# plotting accuracies with max_depth
plt.figure()
plt.title('Effect of different max_depth on accuracy of model')
plt.plot(scores["param_max_depth"], scores["mean_train_score"], label="training acc
plt.plot(scores["param_max_depth"], scores["mean_test_score"], label="test accuracy
plt.xlabel("max_depth")
plt.ylabel("Accuracy")
plt.grid()
plt.legend()
plt.show()
```




```
 Observe that accuracy of test and train data increases till
max_depth of 6, after that training
 data accuracy increases abruptly but test data accuracy saturates
```

Hyper tuning for n_estimators

In [109…

```python
# specify number of folds for k-fold CV
n_folds = 5

# parameters to build the model on
parameters = {'n_estimators': range(100, 1500, 100)}

# instantiate the model (note we are specifying a max_depth)
rf = RandomForestClassifier(max_depth=8)

# fit tree on training data
rf = GridSearchCV(rf, parameters, cv=n_folds, scoring="accuracy",return_train_score
rf.fit(features_train, target_train)

# scores of GridSearch CV
scores = rf.cv_results_
pd.DataFrame(scores).head(2)
```

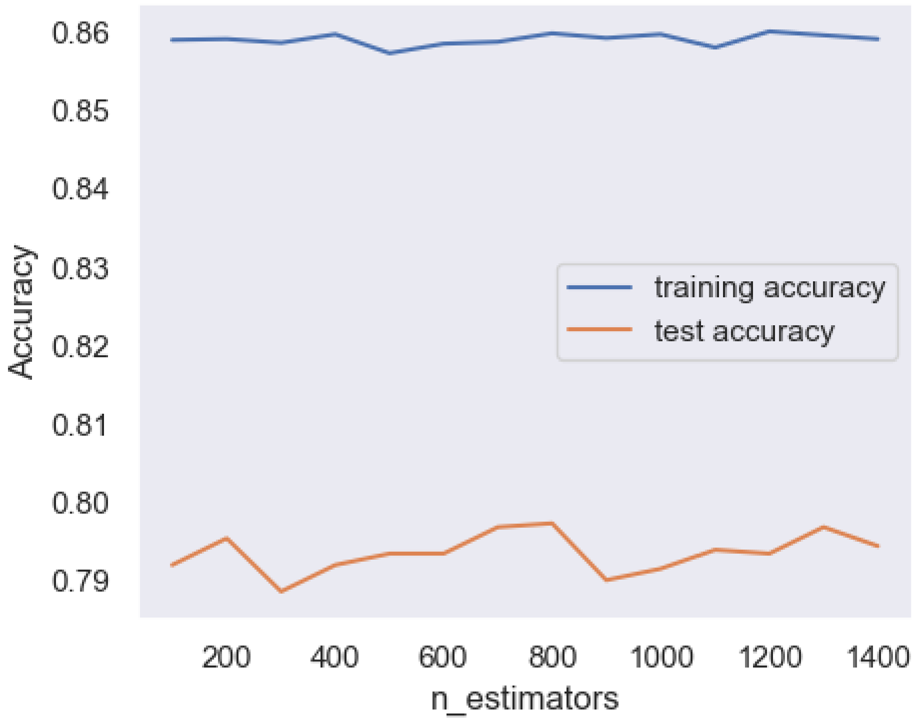
Out[109]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_estimators	params
0	0.222010	0.009280	0.012478	0.006812	100	{'n_estimators' 100
1	0.446943	0.012612	0.020073	0.003816	200	{'n_estimators' 200

2 rows × 21 columns

In [110…

```
# plotting accuracies with n_estimators
plt.figure()
plt.plot(scores["param_n_estimators"], scores["mean_train_score"], label="training
plt.plot(scores["param_n_estimators"], scores["mean_test_score"], label="test accur
plt.xlabel("n_estimators")
plt.ylabel("Accuracy")
plt.grid()
plt.legend()
plt.show()
```



n_estimator is not contributing much towards accuracy of the model and hence we will go with n_estimaor = 300 because beyond that n_estimator VS accuracy curve is almost flat

Hyper tuning for max_features

In [111…

```
# specify number of folds for k-fold CV
n_folds = 5
# parameters to build the model on
parameters = {'max_features': [2,3,5,6,8]}
# instantiate the model
rf = RandomForestClassifier(max_depth=8,n_estimators=300)

# fit tree on training data
rf = GridSearchCV(rf, parameters, cv=n_folds, scoring="accuracy",return_train_score
rf.fit(features_train, target_train)

# scores of GridSearch CV
scores = rf.cv_results_
pd.DataFrame(scores).head(2)
```

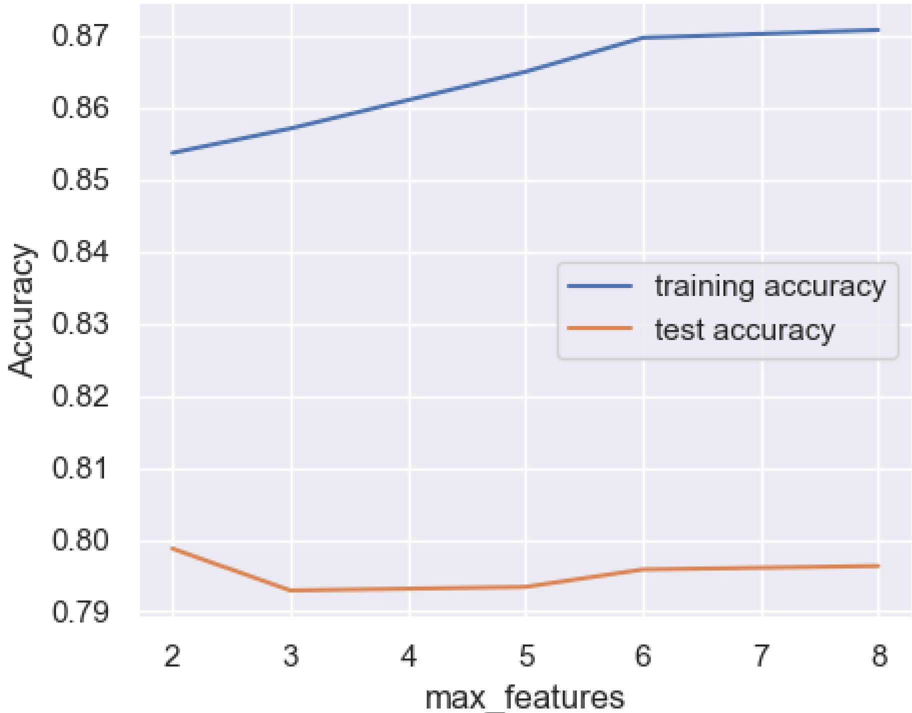
Out[111]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_features	param
0	0.568081	0.040608	0.032208	0.000952	2	{'max_feature:
1	0.614799	0.017854	0.034198	0.006494	3	{'max_feature:

2 rows × 21 columns

In [112…

```
# plotting accuracies with max_features
plt.figure()
plt.plot(scores["param_max_features"], scores["mean_train_score"], label="training
plt.plot(scores["param_max_features"], scores["mean_test_score"], label="test accur
plt.xlabel("max_features")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```




```
On increasing the number of features, the accuracy of training data
is increasing, however,
accuracy of test data is decreasing, This is the case of
oversampling.
```

Hyperparameter tuning using GridSearch - combining all parameters

In [113…

```
from sklearn.model_selection import GridSearchCV
# Define the hyperparameter grid for tuning
param_grid = {
    'max_features': [2,5,6,8],
    'n_estimators': [100, 200, 300, 400],
    'max_depth': [None, 5, 8, 10, 20],
    'min_samples_split': [2, 5, 10],
```

```
    'min_samples_leaf': [1, 2, 4]
}

# Use GridSearchCV for hyperparameter tuning
grid_search_rf_1 = GridSearchCV(rf_classifier, param_grid, cv=5, scoring='accuracy'
grid_search_rf_1.fit(features_train, target_train)

# Get the best hyperparameters
best_params_rf_1 = grid_search_rf_1.best_params_

# Train the Random Forest model with the best hyperparameters
best_rf_classifier = RandomForestClassifier(random_state=42, **best_params_rf_1)
best_rf_classifier.fit(features_train, target_train)

# Make predictions on the test set
target_pred_rf_1 = best_rf_classifier.predict(features_test)

# Evaluate the Random Forest model
print("Random Forest Classification Report:")
print(classification_report(target_test, target_pred_rf_1))
print("Accuracy:", accuracy_score(target_test, target_pred_rf_1))
print('F1 score:',f1_score(target_test, target_pred_rf_1))
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.83      0.83       254
           1       0.83      0.84      0.83       262

    accuracy                           0.83       516
   macro avg       0.83      0.83      0.83       516
weighted avg       0.83      0.83      0.83       516

Accuracy: 0.8313953488372093
F1 score: 0.8342857142857143
```

```
   The model, post hyperparameter tuning, maintains a stable and
balanced performance with
   similar metrics for both classes.
   While the accuracy is slightly lower compared to the initial
model, the improvements in
   precision, recall, and F1-score suggest a more refined and
reliable classifier.
```

Ensemble Learning- Boosting Algorithm

Boosting (GBC - Gradient Boosting)

In [114…

```
from sklearn.ensemble import GradientBoostingClassifier

# Initialize the Gradient Boosting Classifier
gbc_classifier = GradientBoostingClassifier(random_state=42)

# Train the Gradient Boosting model
gbc_classifier = GradientBoostingClassifier(random_state=42)
gbc_classifier.fit(features_train, target_train)

# Make predictions on the test set
target_pred_gbc = gbc_classifier.predict(features_test)
```

Classification Report

In [115…

```
# Evaluate the Gradient Boosting model
print("Gradient Boosting Classifier Classification Report:")
print(classification_report(target_test, target_pred_gbc))
print("Accuracy:", accuracy_score(target_test, target_pred_gbc))
print('F1 score:',f1_score(target_test, target_pred_gbc))
```

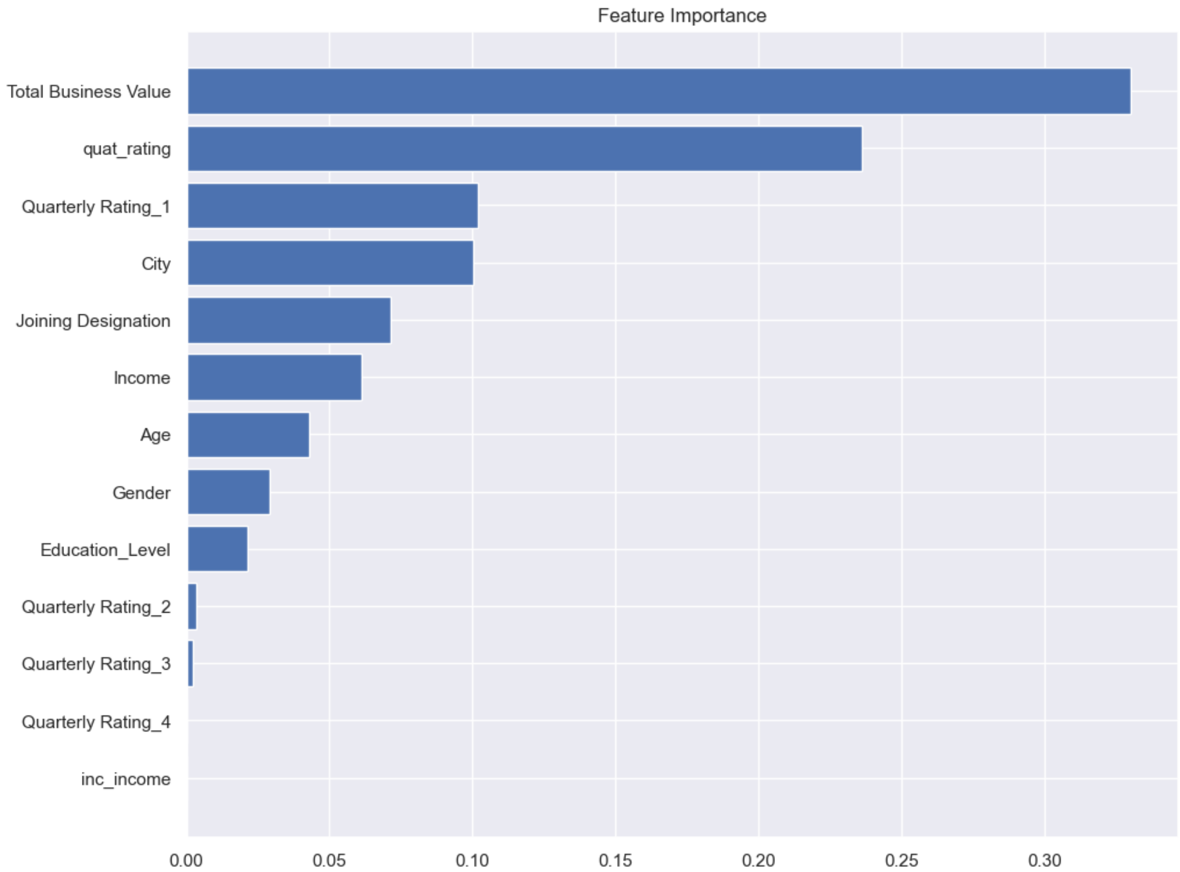
```
Gradient Boosting Classifier Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.78      0.80       254
           1       0.80      0.83      0.81       262

    accuracy                           0.81       516
   macro avg       0.81      0.81      0.81       516
weighted avg       0.81      0.81      0.81       516

Accuracy: 0.8062015503875969
F1 score: 0.8127340823970037
```

1. For class 0, precision is 0.82, recall is 0.78, and the F1-score is 0.80.
2. For class 1, precision is 0.80, recall is 0.83, and the F1-score is 0.81.
3. These metrics indicate a balanced performance for both classes, with slightly higher recall for class 1.
4. The overall accuracy of the Gradient Boosting model is 0.8062, suggesting that it correctly predicted the class labels for approximately 80.6% of the instances.
5. The macro and weighted averages for precision, recall, and F1-score are all 0.81, showing consistent performance across different evaluation metrics.
6. The support values for both classes (254 for class 0, 262 for class 1) indicate a balanced dataset.
7. Compared to the Random Forest model, the Gradient Boosting model has a slightly lower accuracy, precision, recall, and F1-score.

Feature Importance

In [116…

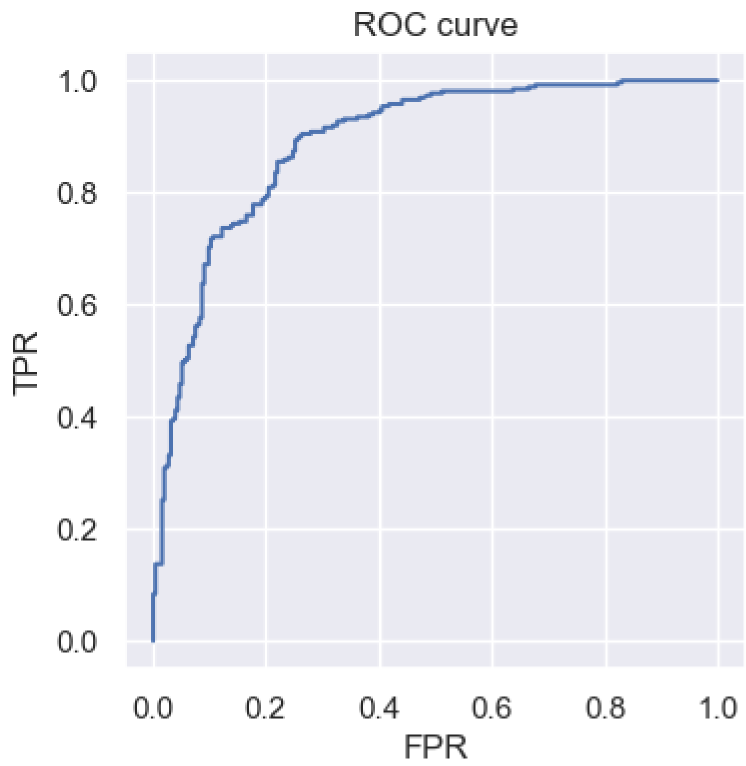
```
feature_importance = gbc_classifier.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + 0.5
fig = plt.figure(figsize=(11, 9))

plt.barh(pos, feature_importance[sorted_idx], align="center")
plt.yticks(pos, np.array(data_4.columns)[sorted_idx])
plt.title("Feature Importance")
plt.show()
```



ROC AUC Curve

In [117…

```
pred_prob2 = gbc_classifier.predict_proba(features_test)
fpr, tpr, thr = roc_curve(target_test,np.ravel(pred_prob2[:,1]))
plt.subplots(figsize=(4,4))
plt.plot(fpr,tpr)
plt.title('ROC curve')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()
print('Area under ROC AUC Curve :', roc_auc_score(target_test,np.ravel(pred_prob2[:
```




```
Area under ROC AUC Curve : 0.8895909719300354
```

1. An AUC score of 0.8896 is relatively high and suggests that model has a strong ability to distinguish between positive and negative classes.

Confusion Matrix

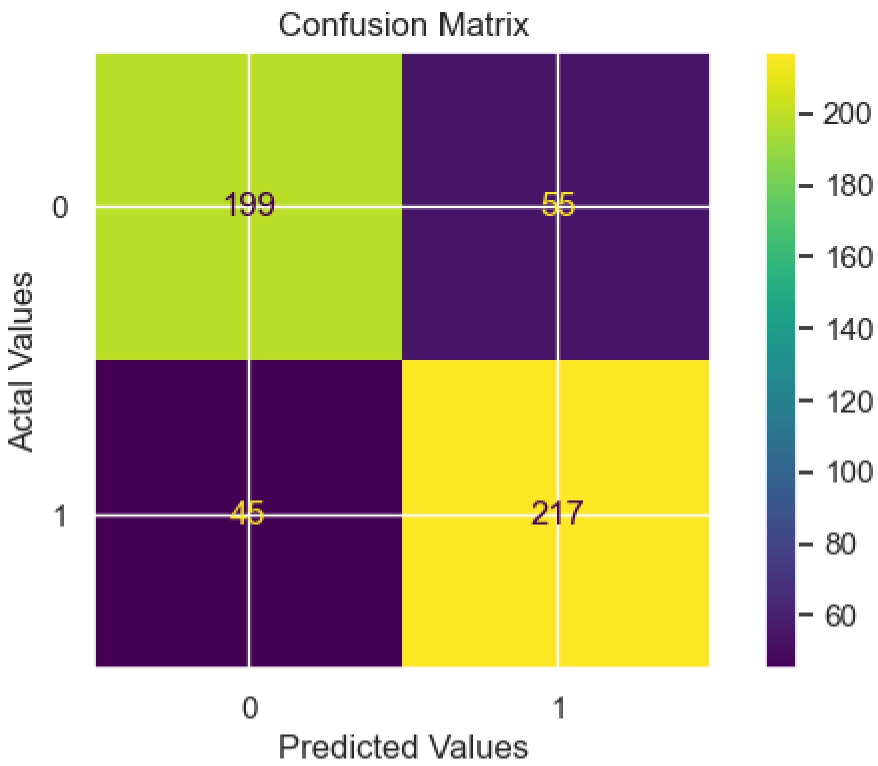
In [118…

```
conf_mat = confusion_matrix(target_test, gbc_classifier.predict(features_test))
print(conf_mat)
```

```
[[199  55]
 [ 45 217]]
```

In [119…

```
#Plotting the confusion matrix
fig, ax = plt.subplots(figsize=(7,4))
cmp = ConfusionMatrixDisplay(conf_mat, display_labels=np.arange(2))
cmp.plot(ax=ax)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



1. True Positives (TP): 217 - Instances correctly predicted as positive (class 1).
2. True Negatives (TN): 199 - Instances correctly predicted as negative (class 0).
3. False Positives (FP): 55 - Instances predicted as positive but actually belong to the negative class.
4. False Negatives (FN): 45 - Instances predicted as negative but actually belong to the positive class.
5. The model has a relatively balanced number of true positives and true negatives, indicating effectiveness in predicting both positive and negative instances.

hyperparameter tuning using GridSearch

In [120…

```
# Define the hyperparameter grid for tuning
param_grid_gbc = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4] }

# Use GridSearchCV for hyperparameter tuning
grid_search_gbc = GridSearchCV(gbc_classifier, param_grid_gbc, cv=5, scoring='accur
grid_search_gbc.fit(features_train, target_train)

# Get the best hyperparameters
best_params_gbc = grid_search_gbc.best_params_

# Train the Gradient Boosting model with the best hyperparameters
best_gbc_classifier = GradientBoostingClassifier(random_state=42, **best_params_gbc
best_gbc_classifier.fit(features_train, target_train)

# Make predictions on the test set
target_pred_gbc_1 = best_gbc_classifier.predict(features_test)
```

```python
# Evaluate the Gradient Boosting model
print("Gradient Boosting Classifier Classification Report with tuning:")
print(classification_report(target_test, target_pred_gbc_1))
print("Gradient Boosting Classifier Accuracy with tuning:", accuracy_score(target_t
```

```
Gradient Boosting Classifier Classification Report with tuning:
              precision    recall  f1-score   support

           0       0.84      0.85      0.85       254
           1       0.85      0.84      0.85       262

    accuracy                           0.85       516
   macro avg       0.85      0.85      0.85       516
weighted avg       0.85      0.85      0.85       516

Gradient Boosting Classifier Accuracy with tuning: 0.8468992248062015
```

```
Hyperparameter tuning has led to improvements in the model's
precision, recall, and F1-score,
resulting in a more balanced and accurate classifier.
1. Precision and recall for both classes (0 and 1) are balanced and
equal at 0.84 and 0.85,
respectively.
2. The F1-score is consistent for both classes at 0.85.
3. The overall accuracy of the model after hyperparameter tuning is
0.8469, indicating that it
correctly predicts the class labels for approximately 84.7% of the
instances.The accuracy has
improved compared to the untuned Gradient Boosting model.
```

Actionable Insights & Recommendations

1. Random Forest Model:
   A. Achieved an accuracy of 84.3% with balanced precision and recall for both classes.
   B. High AUC-ROC score of 0.9055 indicates strong discriminative ability.
   C. Continue utilizing the Random Forest model for its balanced performance.
   D. Monitor its performance on new data to ensure consistent accuracy.
2. Gradient Boosting Model:
   A. Achieved an accuracy of 80.6%, with balanced precision and recall.
   B. AUC-ROC score of 0.8896 indicates good discriminative performance.
   C. Further hyperparameter tuning can potentially improve performance.
3. Hyperparameter-Tuned Gradient Boosting Model:
   A. Improved accuracy to 84.7% with balanced precision and recall.
   B. Achieved consistent F1-scores for both classes.
   C. Continue using the hyperparameter-tuned model for enhanced accuracy.
   D. Assessing its performance over time and on new data is required for better result.

Challenges faced and Insights Drawn:

1. High churn among drivers poses a significant challenge.
2. Acquiring new drivers is costly, impacting overall operational efficiency.
3. Casting a wide net, including individuals without cars, to find new drivers.

4. High driver turnover negatively affects organizational morale.
5. Retaining existing drivers is more cost-effective than recruiting new ones.

Recommendations:

1. Implement targeted strategies to enhance driver retention.
2. Explore incentives or policies to improve driver satisfaction and loyalty.
3. Regularly monitor model performance, especially with changing data patterns.
4. Conduct periodic reevaluations of predictive models.
5. Consider incorporating additional relevant features for a more comprehensive analysis.
6. Explore feedback mechanisms to gather insights directly from drivers.
7. Collaborate with industry stakeholders to address broader challenges in driver retention.
8. Learn from best practices adopted by other ride-sharing platforms.

```
 In summary, using the strengths of Random Forest and the improved Gradient Boosting models,
 along with smart insights, can help Ola create better strategies to keep drivers.
 It's important to regularly check and adjust these strategies as the industry changes to
 ensure long-term success.
```

Questionnaire

1. What percentage of drivers have received a quarterly rating of 5?

ans. ***None*** has received rating of 5.

2. Comment on the correlation between Age and Quarterly Rating.

ans. The correlation value between Age and Quarterly Rating is ***0.171632***. This value is too small, that is, there is very ***low correlation*** between these two features.

3. Name the city which showed the most improvement in Quarterly Rating over the past year

ans. ***C29*** is showing highest improvements in quaterly rating among all cities over past year.

4. Drivers with a Grade of 'A' are more likely to have a higher Total Business Value.

ans. Grade with 4 is more likely to have higher Total Business Value. Grade are represented as 1,2,3,4, and 5. If 1 represents as A, then above statement is wrong. If 5 represents as A, then also above statement is wrong. Therefore, the statement is ***False***

5. If a driver's Quarterly Rating drops significantly, how does it impact their Total Business Value in the subsequent period?

ans. For rating of 4, the total business value is approx 1.115 * 10^6. For rating of 3, total business value is 767804. For rating of 2, total business value is 494266 and for rating of 1, total business value is 83102.9 It can be observed that **as rating decreases by unity, total business value drastically decreased by huge amount**.

**6. From Ola's perspective, which metric should be the primary focus for driver retention?

```
1. ROC AUC
2. Precision
3. Recall
4. F1 Score**
```

ans. The choice of the primary metric for driver retention depends on Ola's specific goals and priorities.Each metric provides a different perspective on model performance, and the decision should align with the business objectives and the cost implications of false positives and false negatives in the context of driver retention.

If the cost of false positives (e.g., unnecessary interventions with drivers) is higher, prioritize *precision*. If the cost of false negatives (missing drivers at risk) is higher, prioritize *recall*. If a balanced approach is desired, *F1 Score* provides a compromise between precision and recall.

It's important to align the metric choice with Ola's business priorities and the specific consequences associated with false positives and false negatives in the context of driver retention.

7. How does the gap in precision and recall affect Ola's relationship with its drivers and customers?

ans. The gap between precision and recall can have significant implications for Ola's relationship with both its drivers and customers.

1. Impact on Drivers:
    A. High Precision, Low Recall: Effect: Ola avoids unnecessary interventions with drivers who may not be at risk of leaving. Impact: Drivers experience fewer disruptions but may lead to missing drivers who are at risk.
    B. Low Precision, High Recall: Effect: Ola captures more drivers at risk of leaving. Impact: Drivers may experience more interventions, potentially leading to dissatisfaction, especially if interventions are unnecessary.
    C. Balanced Approach (Moderate Precision and Recall): Effect: Strikes a balance between minimizing unnecessary interventions and capturing at-risk drivers. Impact: Balances the disruption to drivers with the need to retain those at risk, optimizing driver satisfaction.
2. Impact on Customers:
    A. High Precision, Low Recall: Effect: Fewer disruptions to customer service, as unnecessary interventions are minimized. Impact: Potential loss of drivers may affect service availability and quality, impacting customer satisfaction.
    B. Low Precision, High Recall: Effect: More proactive actions may be taken to address potential driver shortages. Impact: Increased disruptions for customers due to more interventions, potentially affecting customer satisfaction.
    C. Balanced Approach (Moderate Precision and Recall): Effect: Strikes a balance between minimizing unnecessary interventions and addressing driver retention. Impact: Seeks to maintain service quality while addressing potential driver shortages in a more targeted manner.

8. Besides the obvious features like "Number of Rides", which lesser-discussed features might have a strong impact on a driver's Quarterly Rating?

ans. In addition to the obvious features like "Number of Rides," several lesser-discussed features could have a strong impact on a driver's Quarterly Rating are:

1. Average Ride Duration
2. Peak Hour Availability
3. Communication Effectiveness
4. Vehicle Cleanliness
5. Wait Time at Pickup
6. Navigation Skills
7. Safety and Driving Behavior
8. Customer Interaction and Friendliness

9. Will the driver's performance be affected by the City they operate in? (Yes/No)

ans. There is very less correlation (ranging from -0.406516 to 0.474323) between City and all four quarterly ratings. Therefore, driver's performance is **not affected** by the City they opertae in.

10.Analyze any seasonality in the driver's ratings. Do certain times of the year correspond to higher or lower ratings, and why might that be?

a. March 2019 showing the highest rating followed by March 2020. June 2020 showing the lowest rating followed by Dec 2019 and Apr 2019. There is no particular pattern for lowest rating but **highest rating was seen in march month for both year**. Probable reasons may be:

1. In some regions, March is a time for spring breaks for schools and colleges. Increased travel and leisure activities during this time might contribute to positive passenger experiences.
2. March is often associated with the arrival of spring in many regions. Pleasant weather conditions could positively influence passengers' experiences, leading to higher ratings.
3. Is there any promotion activity happen in both years during this time? Any promotions in March could influence both driver behavior and passenger satisfaction.